Sunrise Clinical Manager Documentation

Release 14.3 CU10

Le Yang

March 03, 2016
## Contents

1 Project Management Office .......................... 1
   1.1 Program Goals ........................................ 1
   1.2 Soarian (only) Implementation ...................... 1

2 The Model System ...................................... 5
   2.1 Process & Workflow .................................. 5
   2.2 Configuration ........................................ 11
   2.3 Customizations (ObjectPlus/XA) .................... 16
   2.4 Customizations (MLM) ................................ 16

3 Testing .................................................. 23
   3.1 Methodology .......................................... 23
   3.2 Testing Cycles ........................................ 23

4 Support .................................................. 25
   4.1 Support Overview ..................................... 25
   4.2 Knowledge Base Articles ............................. 25

5 Shared Learning ........................................ 55
   5.1 Automation of Repetitive Tasks ..................... 55
   5.2 An Introduction to MLM .............................. 60
   5.3 HL7 (SCM) Specification Development ............... 63
   5.4 Separation of Presentation and Content ............. 67

6 Miscellaneous .......................................... 73
   6.1 Optimizations ........................................ 73
   6.2 Identification of Research Patients ................ 83

7 Meta ...................................................... 87
   7.1 About these documents .............................. 87
   7.2 Glossary .............................................. 102
1.1 Program Goals

**Improve Patient Safety**
By offering standard, clear documentation and electronic orders that automatically follows the patient; and electronic prompts or alerts that help prevent errors.

**Improve Clinical Outcomes/Demonstrate Quality**
By collecting and analyzing data in a scientifically rigorous manner, St. Vincent Health can use this data confidently and anonymously to make improvements in the way care is delivered.

**Technology Adoption**
A transformational focus will ensure operational readiness in adopting the EHR technology.

**Improve Customer Experience**
Technology adoption will ensure better clinical outcomes and customer service opportunities directly with patients and families.

**Reduce Practice Variation**
By designing a regional based systems using industry best practice and evidence.

**Improve Revenue Capture**
By capturing charges at the point of service through the process of ordering and documenting care.

**Improve Clinical Productivity**
Easier to find and easier to use information improves clinical productivity - and that means more efficient and faster service for patients and care delivery in the long term.

*Section author: Le Yang <le.yang@leidoshealth.com>*

1.2 Soarian (only) Implementation

1.2.1 Milestone: Planning

- How are the new facilities defined in Soarian? Logical definition should match or exceed to financial definitions.
• Location additions/removals.
• Services
• Conversion of charge codes to Soarian CDM.
• Provider NPI’s
• New MRN?
• New Registration?
  – Existing visits will continue to use the same number
  – New visits will create using new number scheme

1.2.2 Milestone: Design/Build

• Add/Remove facilities.
• Add/Remove locations - units and services.
• Add/Remove services.
• Review configurations related facility/location changes
  – “STAT if ER”;
  – print c-cda from discharge tab;
  – reports;
  – other customizations;
  – order auto-status from transfer;
  – location function groups;
  – criteria-based lists/user profiles;
  – location based availability policy;
  – transmission policies;
• Develop charge matrix.
• Build new charge orders and charge order sets.
• Review Soarian related workflows impacts
  – Registrar user profile: CBL
  – orders availability: Registration Change, Patient Status, Discharge, Communication to Registration
  – separate out Stress Center from Indianapolis?
  – bi-directional ADT interface
  – miscellaneous Rx charges
• Review ADT interface specification, per Soarian workflow
  – visit type/care level
  – interface translations
  – other mapping
  – special current state logic
• CPM
  – Special discharge disposition/location SP for non-Soarian facilities
• Clean-up
  – removal of “charge on task” feature.
  – legacy transmission policies (charges)
  – legacy communication policies (charges)
  – legacy charge sets

1.2.3 Milestone: Testing

• Charge testing
• Participate in integrated testing
2.1 Process & Workflow

A workflow consists of an orchestrated and repeatable pattern of business activity enabled by the systematic organization of resources into processes that transform materials, provide services, or process information. It can be depicted as a sequence of operations, declared as work of a person or group, an organization of staff, or one or more simple or complex mechanisms.

Index of workflows

2.1.1 Inpatient Admission and Certification

This document outlines Sunrise implementation of Hospital Inpatient Admission Order and Certification required by CMS.

As a condition of payment for hospital inpatient services under Medicare Part A, section 1814(a) of the Social Security Act requires physician certification of the medical necessity that such services be provided on an inpatient basis. The order to admit as an inpatient (“practitioner order”) is a critical element of the physician certification, and is therefore also required for hospital inpatient coverage and payment under Part A. The physician certification, which includes the practitioner order, is considered along with other documentation in the medical record as evidence that hospital inpatient service(s) were reasonable and necessary. When a physician signs the certification, they are certifying that inpatient hospital services were reasonable and necessary.

Definition of Terms

Certifying Physician In the guideline the Certifying Physician is defined as following:

- the attending physician of record or a physician on call for him or her;
- a surgeon responsible for a major surgical procedure on the beneficiary or a surgeon on call for him or her;
• a dentist functioning as the admitting physician of record or as the surgeon responsible for a major dental procedure;

• and, in the specific case of a non-physician non-dentist admitting practitioner who is licensed by the state and has been granted privileges by the facility, a physician member of the hospital staff who has reviewed the case and who also enters into the record a complete certification statement that specifically contains all of the content elements discussed above.

In Sunrise Clinical Manager, Certifying Physician is defined as users with Provider Type = Physician, or practitioner with the role of Attending who is also a Physician.

**Inpatient Order** An order to formally admit a Medicare beneficiary to the hospital by a physician or other qualified practitioner.

Section author: Le Yang <le.yang@leidoshealth.com>

### 2.1.2 Duplicate Checking Extensions

This document outlines extensions to the Duplicate Checking function available in SCM.

Duplicate checking uses MLMs to check for duplicate orders, including those set as informational orders. Duplicate checking is only performed at the master order item of a complex order for medication orders. For example, it does not alert for each of the rows in a tapering dose medication order. For non-medication Complex orders the master order being placed is compared to child orders that have been generated and master orders that have not yet generated any of the child orders.

Duplicate order checking is designed to:

- Avoid conflicting orders (such as two orders for the same medication).
- Prevent duplication of tests (such as two chest x-rays, outpatient and inpatient).
- Consolidate similar tests (such as H&H and CBC).
- Prevent duplicate consultations (such as two neurology consults for the same reason).

#### Replace matched duplicate order

By setting the duplicate message type to replace or replace even future, newly entered order will supercede matched duplicate orders by discontinuing matched orders. An alert will present to user with matched duplicate orders and replacement actions to be taken similar to other duplicate checking message types. System will proceed to replace matched orders only after the user select Proceed.

If replace is set, only order that is current – orders with significant date/time equal to or less than NOW – will be qualified to trigger the replacement action. This is needed to support workflows where co-existence of multiple orders is allowed as long future date/time is indicated on the additional orders.

If replace even future is set, current order will replace matched duplicate orders, regardless the significant date/time of the order.

Todo
• Is acknowledgement of the alert required before replacement action takes place? *recommend proceed only when acknowledged, replace only when proceeded*

• What is the expectation when the requested care provider does not have the right to discontinue duplicate order? “Request” and “Discontinue” are different rights, while it is rare, the care provider can place new order without able to discontinue old ones. *recommend to not replace*

• What is the discontinue reason and source? *recommend none for source, use duplicate checking additional message for reason*

• What date/time to use to discontinue duplicate orders? Must be greater than now. *recommend 1 minute into the future from now*

• Order set alerting
Use cases

**Food and Nutrition Services**

Food and Nutrition Services orders use the duplicate checking extensions in the following manner:

- The common diet order, **Diet**, will replace **Diet**, **Diet, Bariatric**, and **NPO** using the replace message mode.

- The bariatric diet order, **Diet, Bariatric**, will replace **Diet**, **Diet, Bariatric**, and **NPO** using the replace message mode.

- **NPO** will not replace any Food and Nutrition Services orders.
• Oral supplements and tube feeding orders will continue to function as before i.e. co-exist with diet orders.

The following uses cases are supported by the set up referenced above:

• Newly entered diet order takes precedence and replaces previous (if any) diet orders on chart.
• Preparation for next day surgery/procedure where NPO After Midnight is often ordered the day before.
• Scheduled post-op diet advancements.

**Todo**

Need new duplicate policy? How should the policy be defined? recommend to look back only

---

**Patient Status**

The Patient Status orders – Patient Status, Patient Status - Behavioral Health, Patient Status - Hematology/Oncology – will replace each other with the last entered active order becoming the surviving one, regardless when the order is scheduled to become active.

**Todo**

• Any consideration needed to the Certification order when Patient Status order is replaced?
• Admission order sets current have the rule to dither Patient Status or Resuscitation Status order on the order set if one is always active on the chart. with the replacement of these orders, recommend to remove such restriction on OS so Physicians will have the option to indicate updated Patient Status or Resuscitation Order directly from the OS

---

**Resuscitation Status**

The Resuscitation Status orders will replace itself with the last entered active order becoming the surviving one, regardless when the order is scheduled to become active.

*Section author: Le Yang <le.yang@leidoshealth.com>*

### 2.1.3 Resuscitation Status

This document outlines processes related to Resuscitation Status election.

The Resuscitation Status order is used to indicate elected resuscitation option and intervention options in the case of limited resuscitation. The Resuscitation Status workflow provides the following functions:
Order entry/modify

- Provider must elect one and only one resuscitation status (i.e. Full Resuscitation, Limited Resuscitation, No Resuscitation/Allow Natural Death) before proceeding to submit the order.
- Limited Resuscitation is indicated by selecting at least one prevention.
- Once submitted and activated, order summary line of the Resuscitation Status order is saved to the HEADER1 comment.
- Patient list column Code Status2 is updated with elected resuscitation status: FULL, NO, or LIMITED.

Note:

- HEADER1 comment is a special, system comment that is displayed in the header line.
- Existing limited resuscitation preventions will continue to display in HEADER2 comment unless manually edited/discontinued.

Order discontinue

- HEADER1 comment is replaced with the following texts: Code status not specified.
- Code Status2 patient list column is replaced with the following texts: N/A.
Workflow

2.2 Configuration

Section author: Le Yang <le.yang@leidoshealth.com>

2.2.1 Configure Duplicate Checking Extensions

This document explains configuration steps required to implement Duplicate Checking Extensions. For information on how the process works, see Duplicate Checking Extensions.
Extension: Replacement

To configure Duplicate Checking with Replacement, use the following high-level steps:

1. Determine replacement type

   Two (2) message types are added to the system-supplied HVC-DuplicateMsgType dictionary: replace and replace even future. As explained in the process document:
   - If replace is set, only order that is current – orders with significant date/time equal to or less than NOW – will be qualified to trigger the replacement action. This is needed to support workflows where co-existence of multiple orders is allowed as long future date/time is indicated on the additional orders.
   - If replace even future is set, current order will replace matched duplicate orders, regardless the significant date/time of the order.

2. Set message type and message

   In the Catalog Maintenance window Options list, select Duplicate Checking. Add a new duplicate item and select appropriate message type. In the Additional Message field, enter additional text to be appended to the end of the standard message.

2.2.2 Order Availability Policy

This article explains the nomenclature used to name Order Availability Policy (“OAP”) in consideration to the dithering process. The nomenclature is used to support the multitude of ministries under St. Vincent and the need to configure order catalog items to behave differently at these ministries. A ministry location is defined by a SCM Facility, or the combination of SCM Facility and Location Function Group (e.g. Indianapolis).

What is dithering?

dither (or dithering) is the customization to disable order items on order sets as determined by OAP associated with the order items. The customization updates policy cache several times throughout the day (add time?). Order item availability updates are not reflected on order sets until the next time the policy cache is updated.
OAP model: facility, abbreviation, LFG, and location prefix

<table>
<thead>
<tr>
<th>Abbr</th>
<th>Fac</th>
<th>LFG</th>
<th>Prefix</th>
<th>Ministry</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>AND</td>
<td>A</td>
<td></td>
<td>St. Vincent Anderson Regional</td>
<td>Anderson</td>
</tr>
<tr>
<td>C</td>
<td>CAR</td>
<td>C</td>
<td></td>
<td>St. Vincent Carmel</td>
<td>Carmel</td>
</tr>
<tr>
<td>Y</td>
<td>CLA</td>
<td>CL</td>
<td></td>
<td>St. Vincent Clay</td>
<td>Brazil</td>
</tr>
<tr>
<td>D</td>
<td>DUN</td>
<td>D</td>
<td></td>
<td>St. Vincent Dunn</td>
<td>Bedford</td>
</tr>
<tr>
<td>E</td>
<td>EVV</td>
<td>E</td>
<td></td>
<td>St. Mary’s Health</td>
<td>Evansville</td>
</tr>
<tr>
<td>F</td>
<td>FRK</td>
<td>F</td>
<td></td>
<td>St. Vincent Frankfort</td>
<td>Frankfort</td>
</tr>
<tr>
<td>T</td>
<td>FSH</td>
<td>FH</td>
<td></td>
<td>St. Vincent Fishers</td>
<td>Fishers</td>
</tr>
<tr>
<td>H</td>
<td>HCI</td>
<td>HC</td>
<td></td>
<td>St. Vincent Heart Center</td>
<td>Carmel</td>
</tr>
<tr>
<td>J</td>
<td>JEN</td>
<td>J</td>
<td></td>
<td>St. Vincent Jennings</td>
<td>North Vernon</td>
</tr>
<tr>
<td>K</td>
<td>KOK</td>
<td>K</td>
<td></td>
<td>St. Vincent Kokomo</td>
<td>Kokomo</td>
</tr>
<tr>
<td>M</td>
<td>MER</td>
<td>M</td>
<td></td>
<td>St. Vincent Mercy</td>
<td>Elwood</td>
</tr>
<tr>
<td>R</td>
<td>RAN</td>
<td>R</td>
<td></td>
<td>St. Vincent Randolph</td>
<td>Winchester</td>
</tr>
<tr>
<td>L</td>
<td>SAL</td>
<td>SA</td>
<td></td>
<td>St. Vincent Salem</td>
<td>Salem</td>
</tr>
<tr>
<td>X</td>
<td>SC</td>
<td>S</td>
<td></td>
<td>St. Vincent Stress Center</td>
<td>Indianapolis</td>
</tr>
<tr>
<td>N</td>
<td>SET</td>
<td>SE</td>
<td></td>
<td>St. Vincent Seton Specialty</td>
<td>Indianapolis</td>
</tr>
<tr>
<td>W</td>
<td>STV</td>
<td>W</td>
<td></td>
<td>St. Vincent Women’s Hospital</td>
<td>Indianapolis</td>
</tr>
<tr>
<td>X</td>
<td>STV</td>
<td>I</td>
<td></td>
<td>St. Vincent Indianapolis</td>
<td>Indianapolis</td>
</tr>
<tr>
<td>P</td>
<td>STV</td>
<td>CH</td>
<td></td>
<td>Peyton Manning Children’s Hospital</td>
<td>Indianapolis</td>
</tr>
<tr>
<td>O</td>
<td>WIL</td>
<td>WL</td>
<td></td>
<td>St. Vincent Williamsport</td>
<td>Williamsport</td>
</tr>
<tr>
<td>B</td>
<td>WAR</td>
<td>?</td>
<td></td>
<td>St. Mary’s Warrick Hospital</td>
<td>Boonville</td>
</tr>
</tbody>
</table>

OAP nomenclature and definition (non-Rx)

Below is the grammar of an OAP name:

```
oap_name ::= availability [dither]
availability ::= “All Hidden” | “All Historical Hidden” | fac_list
dither ::= “_” fac_list
fac_list ::= [“A”] [“B”] [“C”] [“D”] [“E”] [“F”] [“H”] [“J”] [“K”] [“L”] [“M”] [“N”] [“O”]
```

See also:

Introduction to Programming Languages/Grammars

How to interpret the OAP name

1. An OAP is configured with the same root properties. Note that default behavior is applied to an order item if no additional details are configured for a facility/location group.

<table>
<thead>
<tr>
<th>Type</th>
<th>Parent Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Behavior</td>
<td>Not Available</td>
</tr>
<tr>
<td>Require Hold/Loc. Checking</td>
<td>deselected</td>
</tr>
</tbody>
</table>
2. A single underscore (_) is optional in the OAP name and is used to designate that the OAP contains dithering definitions.

3. When underscore is omitted from OAP name, it means the order item is searchable using order entry for all named facilities and the order item is available to order from order sets. For facilities not named, the order item is **not** searchable using order entry, however the order item is still available to order from order sets.

4. When underscore is present in the OAP name:
   - portion to the *left* of the underscore (“available” portion) denote above rule should be followed for listed facilities;
   - portion to the *right* of the underscore (“dither” portion) denotes the order item should be dithered/excluded from listed facilities, even from order sets;

5. Special names—All Hidden, All Historical Hidden, ED Only—retain their original meaning and can only to be used in the availability portion of the OAP name.

**How to define an OAP**

All valid facility and location group combinations are defined in policy details. The following production rules are followed in order, meaning once a location matches to one of the rules for a given location, no other rules are processed:

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Session Type</th>
<th>Facility</th>
<th>LFG</th>
<th>Available</th>
<th>Search Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location listed in the “dither” potion</td>
<td>Standard</td>
<td>&lt;facility&gt;</td>
<td>&lt;lfg&gt;</td>
<td>No</td>
<td>Hidden</td>
</tr>
<tr>
<td>Location is All Hidden</td>
<td>Standard</td>
<td>&lt;all facilities&gt;</td>
<td>&lt;all lfg&gt;</td>
<td>Yes</td>
<td>Hidden</td>
</tr>
<tr>
<td>Location is All Historical Hidden</td>
<td>Historical</td>
<td>&lt;All&gt;</td>
<td></td>
<td>No</td>
<td>Hidden</td>
</tr>
<tr>
<td>Discharge</td>
<td></td>
<td>&lt;All&gt;</td>
<td></td>
<td>Yes</td>
<td>Visible</td>
</tr>
<tr>
<td>Location listed in the “available” potion</td>
<td>Standard</td>
<td>&lt;facility&gt;</td>
<td>&lt;lfg&gt;</td>
<td>Yes</td>
<td>Visible</td>
</tr>
</tbody>
</table>
## Sample OAP configurations

<table>
<thead>
<tr>
<th>Sample Policy</th>
<th>Sample Scenario</th>
<th>Carmel</th>
<th>Indianapolis</th>
<th>Kokomo</th>
<th>Evansville</th>
<th>Avail</th>
<th>Search</th>
<th>Avail</th>
<th>Search</th>
<th>Avail</th>
<th>Search</th>
<th>Avail</th>
<th>Search</th>
<th>Avail</th>
<th>Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_E</td>
<td>Bariatric order searchable at Carmel and Indy but need be excluded at Evansville</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All Historical Hidden_K</td>
<td>Car seat study searchable in all facilities but need to be excluded at Kokomo</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All Historical Hidden_E</td>
<td>iSTAT orderable only through order sets but is not available at Evansville</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CEK</td>
<td>A tube feeding order in the formularies of Carmel, Evansville and Kokomo</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## OAP nomenclature and definition (Rx)

Below is the grammar of an OAP name:

\[
\text{oap\_name} \ := \ \text{prefix} \ \text{availability}
\]

\[
\text{prefix} \ := \ \text{“Rx”} \mid \text{“Child”}
\]

\[
\text{availability} \ := \ [\text{“A”}][\text{“B”}][\text{“C”}][\text{“D”}][\text{“E”}][\text{“F”}][\text{“H”}][\text{“J”}][\text{“K”}][\text{“L”}][\text{“M”}][\text{“N”}][\text{“O”}]
\]

See also:

Introduction to Programming Languages/Grammars

### How to interpret the OAP name

1. An OAP is configured with the same root properties. Note that default behavior is applied to an order item if no additional details are configured for a facility/location group.

<table>
<thead>
<tr>
<th>Type</th>
<th>Parent Item or Pre-configured Order Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Behavior</td>
<td>Not Available or blank</td>
</tr>
<tr>
<td>Require Hold/Loc. Checking</td>
<td>deselected</td>
</tr>
</tbody>
</table>

2. When a facility abbreviation is included in the OAP (Rx) name, it means the order item is searchable using order entry for all named facilities in both *Standard* and *Discharge* order session. For facilities not named, the order item is **not** searchable using order entry or order sets.


2.3 Customizations (ObjectPlus/XA)

This section contains the reference material for customizations using SCM’s ObjectPlus/XA technology that exposes parts of the application to enable IT staff to build components that access patient medical records, or extend solutions with data integrated from other systems.

2.4 Customizations (MLM)

(to be completed)

form-func-substitute -r

Site package

Form-based

Functional

A functional MLM usually is generic in nature but contains substantial logic similar to that of a system function.

2.4.1 FORM_FUNC_SUBSTITUTE

Type function

Evoke form called

Purpose

Automatically substitute entries on form-level fields to alternate value as defined by the enterprise during FieldChange event.

Usage

Attach to relevant fields during form configuration.

Explanation

Order and task forms are used to collect data from user. Data entry can be cumbersome, or error prone, or both. This MLM aims to assist these areas by automatically substitute user entries to alternative value as defined by the enterprise.
The accuracy and usefulness of such processing depends on well-defined rules. A rule can be complex in nature however it must be deterministic. Rules will have the following components:

- **field**: Field to apply the rules. Compound fields are appropriate as well, since they can be manipulated similar to simple fields.

- **logic**: Actual embodiment of the rule.

- **substitution**: Substituted data from original user input, after being processed by the rules.

### Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Field</th>
<th>Logic</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weekly frequency on a specific day</td>
<td>Frequency</td>
<td>If simple frequency is entered matching the pattern weekly on day, then change the frequency to weekly on the requested day at 0900 using user-defined entry.</td>
<td>&lt;User Scheduled&gt; every 1 week on day at 0900</td>
</tr>
</tbody>
</table>

### Arguments

**CommunicationObj**

Information about the client, user, care provider, order or task, the form, selected fields from the State-Info object, and the Primary object. Certain fields from the Communication object can be modified and returned to the form.

**FormObj**

Information about the fields on the forms. This object contains field objects for each of the form fields. Certain fields from the Form object can be modified and returned to the form.

**ClientInfoObj**

The Arden ClientInfo object.

### 2.4.2 STV_FUNC_ORDER_CREATE

**Type** function

**Evoke** called

**Purpose**

Create requested order using supplied parameters.

**Explanation**

Calling ObjectPlus/XA static methods to create general, diagnostic, or medication order. This functional MLM has the following features:

- User data are valued before order creation according to types of UDDI’s to be valued.
• Additional data elements specific to diagnostic or medication types are processed, if valued.
• Create templated (child) order if modifier name (and optionally, modifier version) is provided.

Parameters

Arguments

**OrderDefinitionObj**
Order definition object. See *STD_INCLUDE_LIBS* for additional details.

**CareProviderGUID**
GUID of requesting care provider.

**ClientVisitGUID**
GUID of client visit for use in context with the order creation.

**Reason**
Reason texts to display in the order status history. Default to "".

**UserDataObjList**
A list of UserDataObj. Each UserDataObj is processed according to available fields for user input. See *mlm-stv-include-libs* for additional details. Default to empty list.

**Source**
Order source to use when creating the requested order. Default to " ", which will trigger signature requirement for the discontinue action if requesting care provider is different than the current user.

**SessionType**
Standard, Hold, or Discharge. Default to Standard.

**SessionReason**
Reason texts to accompany certain order sessions. Should only be valued for **SessionType** = Hold.

**AvailabilityOverride**
Never, Always, or Prompt. Default to Never.

**SuppressNotification**
Alerts (2), All (7), None (0), SignOnSubmit (4), Warnings (1). Default to None.

**ReferenceOrderGUID**
GUID to order to be used as the base for creating new order. All fields on the reference order form, including system defined, are duplicated to the new order. To override referenced values, explicitly declare new value using the **UserDataObjList** option.

Return

**ErrorMessage**
Cumulative error message during order creation processing. Each entry is enclosed in `<li></li>` tags, expected to render as HTML when displayed with alerts.
2.4.3 STV_FUNC_ORDER_DISCONTINUE

Type function
Evoke called

Purpose

Discontinue requested order using supplied date/time, care provider, and reason.

Explanation

Persistent order object is first retrieved using the provided GUID. Once the ORDEROBJ (Order Persistent Object) is retrieved, the Discontinue() function is called using supplied or default parameters.

Parameters

Arguments

CareProviderGUID
GUID of care provider initiating the discontinue request

OrderGUID
GUID of order to discontinue

Source
Order source to use when discontinuing the requested order. Default to "", which will trigger signature requirement for the discontinue action if requesting care provider is different than the current user.

Reason
Reason texts to display in the order status history. Default to "".

DiscontinueDate
Date to use when discontinuing the requested order in MM/DD/YYYY format. Default to now + 1 minute, since the discontinue date/time has to be greater than the current date/time.

DiscontinueTime
Time to use when discontinuing the requested order in HH:mm format. Default to now + 1 minute, since the discontinue date/time has to be greater than the current date/time.

ErrorMessage
Cumulative error message during order discontinue processing.
Utility/System

2.4.4 STDINCLUDELIBS

Type system
Evoke none

Purpose

Similar to the system included libraries, include common, site-specific definitions and object declarations.

Explanation

There is no separate MLM. Contents are instead included in the actual STD system library.

Usage

Non-standalone, designed to include in other MLM's. Sample syntax:

```plaintext
stv_standard_libs := mlm 'STD_INCLUDE_LIBS' ;
include std_standard_libs ;
```

Object Definitions

All fields are strings unless otherwise noted.

OrderFlagObj

All fields are Boolean types, expecting true or false.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsPRN</td>
<td></td>
</tr>
<tr>
<td>IsBatched</td>
<td></td>
</tr>
<tr>
<td>IsReadOnly</td>
<td></td>
</tr>
<tr>
<td>NoActionOnDischarge</td>
<td></td>
</tr>
<tr>
<td>NoActionOnTransfer</td>
<td></td>
</tr>
<tr>
<td>ScheduleFromWorklist</td>
<td></td>
</tr>
<tr>
<td>SuppressTaskGeneration</td>
<td></td>
</tr>
<tr>
<td>UserWakingHours</td>
<td></td>
</tr>
</tbody>
</table>
OrderDefinitionObj

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of order item</td>
</tr>
<tr>
<td>RequestedDate</td>
<td>YYYY-MM-DD</td>
</tr>
<tr>
<td>RequestedTime</td>
<td>Coded time or HH:mm</td>
</tr>
<tr>
<td>Frequency</td>
<td>Coded frequency</td>
</tr>
<tr>
<td>PRNReason</td>
<td></td>
</tr>
<tr>
<td>ReviewDate</td>
<td></td>
</tr>
<tr>
<td>ReviewTime</td>
<td></td>
</tr>
<tr>
<td>SpecialInstructions</td>
<td></td>
</tr>
<tr>
<td>StopAfter</td>
<td></td>
</tr>
<tr>
<td>StopDate</td>
<td></td>
</tr>
<tr>
<td>StopTime</td>
<td></td>
</tr>
<tr>
<td>TransferLocation</td>
<td></td>
</tr>
<tr>
<td>TransportMethod</td>
<td></td>
</tr>
<tr>
<td>FlagObj</td>
<td>OrderFlagObj</td>
</tr>
<tr>
<td>TypeCode</td>
<td>general, medication, or diagnostic</td>
</tr>
<tr>
<td>ModifierName</td>
<td></td>
</tr>
<tr>
<td>ModifierVersion</td>
<td></td>
</tr>
</tbody>
</table>

UserDataObj

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataItem</td>
<td>UDDI name</td>
</tr>
<tr>
<td>Value</td>
<td></td>
</tr>
</tbody>
</table>
3.1 Methodology

3.2 Testing Cycles

3.2.1 Regression testing

In Regression Testing, the goal..
4.1 Support Overview

Technical support is often subdivided into tiers, or levels, in order to better serve a business or customer base. The number of levels a business uses to organize their technical support group is dependent on a business’ needs regarding their ability to sufficiently serve their customers or users. The reason for providing a multi-tiered support system instead of one general support group is to provide the best possible service in the most efficient possible manner. Success of the organizational structure is dependent on the technicians’ understanding of their level of responsibility and commitments, their customer response time commitments, and when to appropriately escalate an issue and to which level. A common support structure revolves around a three-tiered technical support system.

4.1.1 Tier 1

Tier I (or Level 1, abbreviated as T1 or L1) is the initial support level responsible for basic customer issues. It is synonymous with first-line support, level 1 support, front-end support, support line 1, and various other headings denoting basic level technical support functions. The first job of a Tier I specialist is to gather the customer’s information and to determine the customer’s issue by analyzing the symptoms and figuring out the underlying problem. When analyzing the symptoms, it is important for the technician to identify what the customer is trying to accomplish so that time is not wasted on “attempting to solve a symptom instead of a problem.”

4.2 Knowledge Base Articles

This page contains the index of all Knowledge Base articles, known as KBs. KB numbers are assigned by the KB editors, and once assigned are never changed. The revision control history of the KB texts represent their historical record.

4.2.1 KB Guideline sample

PEP 1

Title PEP Purpose and Guidelines
What is a PEP?

PEP stands for Python Enhancement Proposal. A PEP is a design document providing information to the Python community, or describing a new feature for Python or its processes or environment. The PEP should provide a concise technical specification of the feature and a rationale for the feature.

We intend PEPs to be the primary mechanisms for proposing major new features, for collecting community input on an issue, and for documenting the design decisions that have gone into Python. The PEP author is responsible for building consensus within the community and documenting dissenting opinions.

Because the PEPs are maintained as text files in a versioned repository, their revision history is the historical record of the feature proposal.

PEP Types

There are three kinds of PEP:

1. A Standards Track PEP describes a new feature or implementation for Python. It may also describe an interoperability standard that will be supported outside the standard library for current Python versions before a subsequent PEP adds standard library support in a future version.

2. An Informational PEP describes a Python design issue, or provides general guidelines or information to the Python community, but does not propose a new feature. Informational PEPs do not necessarily represent a Python community consensus or recommendation, so users and implementers are free to ignore Informational PEPs or follow their advice.

3. A Process PEP describes a process surrounding Python, or proposes a change to (or an event in) a process. Process PEPs are like Standards Track PEPs but apply to areas other than the Python language itself. They may propose an implementation, but not to Python’s codebase; they often require community consensus; unlike Informational PEPs, they are more than recommendations, and users are typically not free to ignore them. Examples include procedures, guidelines, changes to the decision-making process, and changes to the tools or environment used in Python development. Any meta-PEP is also considered a Process PEP.

---

1 This historical record is available by the normal hg commands for retrieving older revisions, and can also be browsed via HTTP here: http://hg.python.org/peps/
PEP Workflow

Python’s BDFL

There are several references in this PEP to the “BDFL”. This acronym stands for “Benevolent Dictator for Life” and refers to Guido van Rossum, the original creator of, and the final design authority for, the Python programming language.

PEP Editors

The PEP editors are individuals responsible for managing the administrative and editorial aspects of the PEP workflow (e.g. assigning PEP numbers and changing their status). See PEP Editor Responsibilities & Workflow for details. The current editors are:

- Chris Angelico
- Anthony Baxter
- Georg Brandl
- Brett Cannon
- David Goodger
- Jesse Noller
- Berker Peksag
- Guido van Rossum
- Barry Warsaw

PEP editorship is by invitation of the current editors. The address <peps@python.org> is a mailing list for contacting the PEP editors. All email related to PEP administration (such as requesting a PEP number or providing an updated version of a PEP for posting) should be sent to this address (no cross-posting please).

Submitting a PEP

The PEP process begins with a new idea for Python. It is highly recommended that a single PEP contain a single key proposal or new idea. Small enhancements or patches often don’t need a PEP and can be injected into the Python development workflow with a patch submission to the Python issue tracker. The more focused the PEP, the more successful it tends to be. The PEP editors reserve the right to reject PEP proposals if they appear too unfocused or too broad. If in doubt, split your PEP into several well-focused ones.

Each PEP must have a champion – someone who writes the PEP using the style and format described below, shepherds the discussions in the appropriate forums, and attempts to build community consensus around the idea. The PEP champion (a.k.a. Author) should first attempt to ascertain whether the idea is PEP-able. Posting to the comp.lang.python newsgroup (a.k.a. python-list@python.org mailing list) or the python-ideas mailing list is the best way to go about this.
Vetting an idea publicly before going as far as writing a PEP is meant to save the potential author time. Many ideas have been brought forward for changing Python that have been rejected for various reasons. Asking the Python community first if an idea is original helps prevent too much time being spent on something that is guaranteed to be rejected based on prior discussions (searching the internet does not always do the trick). It also helps to make sure the idea is applicable to the entire community and not just the author. Just because an idea sounds good to the author does not mean it will work for most people in most areas where Python is used.

Once the champion has asked the Python community as to whether an idea has any chance of acceptance, a draft PEP should be presented to python-ideas. This gives the author a chance to flesh out the draft PEP to make properly formatted, of high quality, and to address initial concerns about the proposal.

Following a discussion on python-ideas, the proposal should be sent as a draft PEP to the PEP editors <peps@python.org>. The draft must be written in PEP style as described below, else it will be sent back without further regard until proper formatting rules are followed (although minor errors will be corrected by the editors).

If the PEP editors approve, they will assign the PEP a number, label it as Standards Track, Informational, or Process, give it status “Draft”, and create and check-in the initial draft of the PEP. The PEP editors will not unreasonably deny a PEP. Reasons for denying PEP status include duplication of effort, being technically unsound, not providing proper motivation or addressing backwards compatibility, or not in keeping with the Python philosophy. The BDFL can be consulted during the approval phase, and is the final arbiter of the draft’s PEP-ability.

Developers with hg push privileges for the PEP repository may claim PEP numbers directly by creating and committing a new PEP. When doing so, the developer must handle the tasks that would normally be taken care of by the PEP editors (see PEP Editor Responsibilities & Workflow). This includes ensuring the initial version meets the expected standards for submitting a PEP. Alternately, even developers may choose to submit PEPs through the PEP editors. When doing so, let the PEP editors know you have hg push privileges and they can guide you through the process of updating the PEP repository directly.

As updates are necessary, the PEP author can check in new versions if they (or a collaborating developer) have hg push privileges, or else they can email new PEP versions to the PEP editors for publication.

After a PEP number has been assigned, a draft PEP may be discussed further on python-ideas (getting a PEP number assigned early can be useful for ease of reference, especially when multiple draft PEPs are being considered at the same time). Eventually, all Standards Track PEPs must be sent to the python-dev list for review as described in the next section.

Standards Track PEPs consist of two parts, a design document and a reference implementation. It is generally recommended that at least a prototype implementation be co-developed with the PEP, as ideas that sound good in principle sometimes turn out to be impractical when subjected to the test of implementation.

PEP authors are responsible for collecting community feedback on a PEP before submitting it for review. However, wherever possible, long open-ended discussions on public mailing lists should be avoided. Strategies to keep the discussions efficient include: setting up a separate SIG mailing list for the topic, having the PEP author accept private comments in the early design phases, setting up a wiki page, etc. PEP authors should use their discretion here.
PEP Review & Resolution

Once the authors have completed a PEP, they may request a review for style and consistency from the PEP editors. However, the content and final acceptance of the PEP must be requested of the BDFL, usually via an email to the python-dev mailing list. PEPs are reviewed by the BDFL and his chosen consultants, who may accept or reject a PEP or send it back to the author(s) for revision. For a PEP that is predetermined to be acceptable (e.g., it is an obvious win as-is and/or its implementation has already been checked in) the BDFL may also initiate a PEP review, first notifying the PEP author(s) and giving them a chance to make revisions.

The final authority for PEP approval is the BDFL. However, whenever a new PEP is put forward, any core developer that believes they are suitably experienced to make the final decision on that PEP may offer to serve as the BDFL’s delegate (or “PEP czar”) for that PEP. If their self-nomination is accepted by the other core developers and the BDFL, then they will have the authority to approve (or reject) that PEP. This process happens most frequently with PEPs where the BDFL has granted in principle approval for something to be done, but there are details that need to be worked out before the PEP can be accepted.

If the final decision on a PEP is to be made by a delegate rather than directly by the BDFL, this will be recorded by including the “BDFL-Delegate” header in the PEP.

PEP review and resolution may also occur on a list other than python-dev (for example, distutils-sig for packaging related PEPs that don’t immediately affect the standard library). In this case, the “Discussions-To” heading in the PEP will identify the appropriate alternative list where discussion, review and pronouncement on the PEP will occur.

For a PEP to be accepted it must meet certain minimum criteria. It must be a clear and complete description of the proposed enhancement. The enhancement must represent a net improvement. The proposed implementation, if applicable, must be solid and must not complicate the interpreter unduly. Finally, a proposed enhancement must be “pythonic” in order to be accepted by the BDFL. (However, “pythonic” is an imprecise term; it may be defined as whatever is acceptable to the BDFL. This logic is intentionally circular.) See PEP 2 for standard library module acceptance criteria.

Once a PEP has been accepted, the reference implementation must be completed. When the reference implementation is complete and incorporated into the main source code repository, the status will be changed to “Final”.

A PEP can also be assigned status “Deferred”. The PEP author or an editor can assign the PEP this status when no progress is being made on the PEP. Once a PEP is deferred, a PEP editor can re-assign it to draft status.

A PEP can also be “Rejected”. Perhaps after all is said and done it was not a good idea. It is still important to have a record of this fact. The “Withdrawn” status is similar - it means that the PEP author themselves has decided that the PEP is actually a bad idea, or has accepted that a competing proposal is a better alternative.

When a PEP is Accepted, Rejected or Withdrawn, the PEP should be updated accordingly. In addition to updating the status field, at the very least the Resolution header should be added with a link to the relevant post in the python-dev mailing list archives.

PEPs can also be superseded by a different PEP, rendering the original obsolete. This is intended for Informational PEPs, where version 2 of an API can replace version 1.

---

2 PEP 2, Procedure for Adding New Modules, Faassen (http://www.python.org/dev/peps/pep-0002)
The possible paths of the status of PEPs are as follows:

Some Informational and Process PEPs may also have a status of “Active” if they are never meant to be completed. E.g. PEP 1 (this PEP).

PEP Maintenance

In general, Standards track PEPs are no longer modified after they have reached the Final state. Once a PEP has been completed, the Language and Standard Library References become the formal documentation of the expected behavior.

Informational and Process PEPs may be updated over time to reflect changes to development practices and other details. The precise process followed in these cases will depend on the nature and purpose of the PEP being updated.

What belongs in a successful PEP?

Each PEP should have the following parts:

1. Preamble – RFC 822 style headers containing meta-data about the PEP, including the PEP number, a short descriptive title (limited to a maximum of 44 characters), the names, and optionally the contact info for each author, etc.


3. Copyright/public domain – Each PEP must either be explicitly labeled as placed in the public domain (see this PEP as an example) or licensed under the Open Publication License.

4. Specification – The technical specification should describe the syntax and semantics of any new language feature. The specification should be detailed enough to allow competing, interoperable implementations for at least the current major Python platforms (CPython, Jython, IronPython, PyPy).

5. Motivation – The motivation is critical for PEPs that want to change the Python language. It should clearly explain why the existing language specification is inadequate to address the problem that the PEP solves. PEP submissions without sufficient motivation may be rejected outright.

6. Rationale – The rationale fleshes out the specification by describing what motivated the design and why particular design decisions were made. It should describe alternate designs that were considered and related work, e.g. how the feature is supported in other languages.

   The rationale should provide evidence of consensus within the community and discuss important objections or concerns raised during discussion.

7. Backwards Compatibility – All PEPs that introduce backwards incompatibilities must include a section describing these incompatibilities and their severity. The PEP must explain how the author proposes to deal with these incompatibilities. PEP submissions without a sufficient backwards compatibility treatise may be rejected outright.

8. Reference Implementation – The reference implementation must be completed before any PEP is given status “Final”, but it need not be completed before the PEP is accepted. While there is merit to the approach of reaching consensus on the specification and rationale before writing code, the
principle of “rough consensus and running code” is still useful when it comes to resolving many discussions of API details.

The final implementation must include test code and documentation appropriate for either the Python language reference or the standard library reference.

**PEP Formats and Templates**

There are two PEP formats available to authors: plaintext and reStructuredText. Both are UTF-8-encoded text files.

Plaintext PEPs are written with minimal structural markup that adheres to a rigid style. PEP 9 contains instructions and a template you can use to get started writing your plaintext PEP.

ReStructuredText PEPs allow for rich markup that is still quite easy to read, but results in much better-looking and more functional HTML. PEP 12 contains instructions and a template for reStructuredText PEPs.

There is a Python script that converts both styles of PEPs to HTML for viewing on the web. Parsing and conversion of plaintext PEPs is self-contained within the script. reStructuredText PEPs are parsed and converted by Docutils code called from the script.

**PEP Header Preamble**

Each PEP must begin with an RFC 822 style header preamble. The headers must appear in the following order. Headers marked with “*” are optional and are described below. All other headers are required.

```plaintext
PEP: <pep number>
Title: <pep title>
Version: <version string>
Last-Modified: <date string>
Author: <list of authors' real names and optionally, email addr>
* BDFL-Delegate: <PEP czar's real name>
* Discussions-To: <email address>
Status: <Draft | Active | Accepted | Deferred | Rejected | Withdrawn | Final | Superseded>
Type: <Standards Track | Informational | Process>
* Content-Type: <text/plain | text/x-rst>
* Requires: <pep numbers>
Created: <date created on, in dd-mmm-yyyy format>
* Python-Version: <version number>
Post-History: <dates of postings to python-list and python-dev>
* Replaces: <pep number>
* Superseded-By: <pep number>
* Resolution: <url>
```

---

3 PEP 9, Sample Plaintext PEP Template, Warsaw (http://www.python.org/dev/peps/pep-0009)
4 PEP 12, Sample reStructuredText PEP Template, Goodger, Warsaw (http://www.python.org/dev/peps/pep-0012)
5 The script referred to here is pep2pyramid.py, the successor to pep2html.py, both of which live in the same directory in the hg repo as the PEPs themselves. Try pep2html.py --help for details. The URL for viewing PEPs on the web is http://www.python.org/dev/peps/.
The Author header lists the names, and optionally the email addresses of all the authors/owners of the PEP. The format of the Author header value must be

```
Random J. User <address@dom.ain>
```

if the email address is included, and just

```
Random J. User
```

if the address is not given. For historical reasons the format “address@dom.ain (Random J. User)” may appear in a PEP, however new PEPs must use the mandated format above, and it is acceptable to change to this format when PEPs are updated.

If there are multiple authors, each should be on a separate line following RFC 2822 continuation line conventions. Note that personal email addresses in PEPs will be obscured as a defense against spam harvesters.

The BDFL-Delegate field is used to record cases where the final decision to approve or reject a PEP rests with someone other than the BDFL. (The delegate’s email address is currently omitted due to a limitation in the email address masking for reStructuredText PEPs)

Note: The Resolution header is required for Standards Track PEPs only. It contains a URL that should point to an email message or other web resource where the pronouncement about the PEP is made.

For a PEP where final pronouncement will be made on a list other than python-dev, a Discussions-To header will indicate the mailing list or URL where the pronouncement will occur. A temporary Discussions-To header may also be used when a draft PEP is being discussed prior to submission for pronouncement. No Discussions-To header is necessary if the PEP is being discussed privately with the author, or on the python-list, python-ideas or python-dev mailing lists. Note that email addresses in the Discussions-To header will not be obscured.

The Type header specifies the type of PEP: Standards Track, Informational, or Process.

The format of a PEP is specified with a Content-Type header. The acceptable values are “text/plain” for plaintext PEPs (see PEP 9) and “text/x-rst” for reStructuredText PEPs (see PEP 12). Plaintext (“text/plain”) is the default if no Content-Type header is present.

The Created header records the date that the PEP was assigned a number, while Post-History is used to record the dates of when new versions of the PEP are posted to python-list and/or python-dev. Both headers should be in dd-mmm-yyyy format, e.g. 14-Aug-2001.

Standards Track PEPs will typically have a Python-Version header which indicates the version of Python that the feature will be released with. Standards Track PEPs without a Python-Version header indicate interoperability standards that will initially be supported through external libraries and tools, and then supplemented by a later PEP to add support to the standard library. Informational and Process PEPs do not need a Python-Version header.

PEPs may have a Requires header, indicating the PEP numbers that this PEP depends on.

PEPs may also have a Superseded-By header indicating that a PEP has been rendered obsolete by a later document; the value is the number of the PEP that replaces the current document. The newer PEP must have a Replaces header containing the number of the PEP that it rendered obsolete.
Auxiliary Files

PEPs may include auxiliary files such as diagrams. Such files must be named `pep-XXXX-Y.ext`, where “XXXX” is the PEP number, “Y” is a serial number (starting at 1), and “ext” is replaced by the actual file extension (e.g. “png”).

Reporting PEP Bugs, or Submitting PEP Updates

How you report a bug, or submit a PEP update depends on several factors, such as the maturity of the PEP, the preferences of the PEP author, and the nature of your comments. For the early draft stages of the PEP, it’s probably best to send your comments and changes directly to the PEP author. For more mature, or finished PEPs you may want to submit corrections to the Python issue tracker so that your changes don’t get lost. If the PEP author is a Python developer, assign the bug/patch to them, otherwise assign it to a PEP editor.

When in doubt about where to send your changes, please check first with the PEP author and/or a PEP editor. PEP authors with hg push privileges for the PEP repository can update the PEPs themselves by using “hg push” to submit their changes.

Transferring PEP Ownership

It occasionally becomes necessary to transfer ownership of PEPs to a new champion. In general, it is preferable to retain the original author as a co-author of the transferred PEP, but that’s really up to the original author. A good reason to transfer ownership is because the original author no longer has the time or interest in updating it or following through with the PEP process, or has fallen off the face of the ‘net (i.e. is unreachable or not responding to email). A bad reason to transfer ownership is because the author doesn’t agree with the direction of the PEP. One aim of the PEP process is to try to build consensus around a PEP, but if that’s not possible, an author can always submit a competing PEP.

If you are interested in assuming ownership of a PEP, send a message asking to take over, addressed to both the original author and the PEP editors `<peps@python.org>`. If the original author doesn’t respond to email in a timely manner, the PEP editors will make a unilateral decision (it’s not like such decisions can’t be reversed :).

PEP Editor Responsibilities & Workflow

A PEP editor must subscribe to the `<peps@python.org>` list. All correspondence related to PEP administration should be sent (or forwarded) to `<peps@python.org>` (but please do not cross-post!).

For each new PEP that comes in an editor does the following:

- Read the PEP to check if it is ready: sound and complete. The ideas must make technical sense, even if they don’t seem likely to be accepted.
- The title should accurately describe the content.
- Edit the PEP for language (spelling, grammar, sentence structure, etc.), markup (for reST PEPs), code style (examples should match PEP 8 & 7).
If the PEP isn’t ready, an editor will send it back to the author for revision, with specific instructions.

Once the PEP is ready for the repository, a PEP editor will:

- Assign a PEP number (almost always just the next available number, but sometimes it’s a special/joke number, like 666 or 3141). (Clarification: For Python 3, numbers in the 3000s were used for Py3k-specific proposals. But now that all new features go into Python 3 only, the process is back to using numbers in the 100s again. Remember that numbers below 100 are meta-PEPs.)

- Add the PEP to a local clone of the PEP repository. For mercurial workflow instructions, follow The Python Developers Guide

The mercurial repo for the peps is:

http://hg.python.org/peps/

- Run ./genpepindex.py and ./pep2html.py <PEP Number> to ensure they are generated without errors. If either triggers errors, then the web site will not be updated to reflect the PEP changes.

- Commit and push the new (or updated) PEP

- Monitor python.org to make sure the PEP gets added to the site properly. If it fails to appear, running make will build all of the current PEPs. If any of these are triggering errors, they must be corrected before any PEP will update on the site.

- Send email back to the PEP author with next steps (post to python-list & -dev).

Updates to existing PEPs also come in to peps@python.org. Many PEP authors are not Python committers yet, so PEP editors do the commits for them.

Many PEPs are written and maintained by developers with write access to the Python codebase. The PEP editors monitor the python-checkins list for PEP changes, and correct any structure, grammar, spelling, or markup mistakes they see.

PEP editors don’t pass judgment on PEPs. They merely do the administrative & editorial part (which is generally a low volume task).

Resources:

- Index of Python Enhancement Proposals
- Following Python’s Development
- Python Developer’s Guide
- Frequently Asked Questions for Developers

References and Footnotes

4.2.2 KB Guideline

PEP 1

Title PEP Purpose and Guidelines
What is a PEP?

PEP stands for Python Enhancement Proposal. A PEP is a design document providing information to the Python community, or describing a new feature for Python or its processes or environment. The PEP should provide a concise technical specification of the feature and a rationale for the feature.

We intend PEPs to be the primary mechanisms for proposing major new features, for collecting community input on an issue, and for documenting the design decisions that have gone into Python. The PEP author is responsible for building consensus within the community and documenting dissenting opinions.

Because the PEPs are maintained as text files in a versioned repository, their revision history is the historical record of the feature proposal.

PEP Types

There are three kinds of PEP:

1. A **Standards Track** PEP describes a new feature or implementation for Python. It may also describe an interoperability standard that will be supported outside the standard library for current Python versions before a subsequent PEP adds standard library support in a future version.

2. An **Informational** PEP describes a Python design issue, or provides general guidelines or information to the Python community, but does not propose a new feature. Informational PEPs do not necessarily represent a Python community consensus or recommendation, so users and implementers are free to ignore Informational PEPs or follow their advice.

3. A **Process** PEP describes a process surrounding Python, or proposes a change to (or an event in) a process. Process PEPs are like Standards Track PEPs but apply to areas other than the Python language itself. They may propose an implementation, but not to Python’s codebase; they often require community consensus; unlike Informational PEPs, they are more than recommendations, and users are typically not free to ignore them. Examples include procedures, guidelines, changes to the decision-making process, and changes to the tools or environment used in Python development. Any meta-PEP is also considered a Process PEP.

---

1. This historical record is available by the normal hg commands for retrieving older revisions, and can also be browsed via HTTP here: http://hg.python.org/peps/
PEP Workflow

Python’s BDFL

There are several reference in this PEP to the “BDFL”. This acronym stands for “Benevolent Dictator for Life” and refers to Guido van Rossum, the original creator of, and the final design authority for, the Python programming language.

PEP Editors

The PEP editors are individuals responsible for managing the administrative and editorial aspects of the PEP workflow (e.g. assigning PEP numbers and changing their status). See PEP Editor Responsibilities & Workflow for details. The current editors are:

- Chris Angelico
- Anthony Baxter
- Georg Brandl
- Brett Cannon
- David Goodger
- Jesse Noller
- Berker Peksag
- Guido van Rossum
- Barry Warsaw

PEP editorship is by invitation of the current editors. The address <peps@python.org> is a mailing list for contacting the PEP editors. All email related to PEP administration (such as requesting a PEP number or providing an updated version of a PEP for posting) should be sent to this address (no cross-posting please).

Submitting a PEP

The PEP process begins with a new idea for Python. It is highly recommended that a single PEP contain a single key proposal or new idea. Small enhancements or patches often don’t need a PEP and can be injected into the Python development workflow with a patch submission to the Python issue tracker. The more focused the PEP, the more successful it tends to be. The PEP editors reserve the right to reject PEP proposals if they appear too unfocused or too broad. If in doubt, split your PEP into several well-focused ones.

Each PEP must have a champion – someone who writes the PEP using the style and format described below, shepherds the discussions in the appropriate forums, and attempts to build community consensus around the idea. The PEP champion (a.k.a. Author) should first attempt to ascertain whether the idea is PEP-able. Posting to the comp.lang.python newsgroup (a.k.a. python-list@python.org mailing list) or the python-ideas mailing list is the best way to go about this.
Vetting an idea publicly before going as far as writing a PEP is meant to save the potential author time. Many ideas have been brought forward for changing Python that have been rejected for various reasons. Asking the Python community first if an idea is original helps prevent too much time being spent on something that is guaranteed to be rejected based on prior discussions (searching the internet does not always do the trick). It also helps to make sure the idea is applicable to the entire community and not just the author. Just because an idea sounds good to the author does not mean it will work for most people in most areas where Python is used.

Once the champion has asked the Python community as to whether an idea has any chance of acceptance, a draft PEP should be presented to python-ideas. This gives the author a chance to flesh out the draft PEP to make properly formatted, of high quality, and to address initial concerns about the proposal.

Following a discussion on python-ideas, the proposal should be sent as a draft PEP to the PEP editors <peps@python.org>. The draft must be written in PEP style as described below, else it will be sent back without further regard until proper formatting rules are followed (although minor errors will be corrected by the editors).

If the PEP editors approve, they will assign the PEP a number, label it as Standards Track, Informational, or Process, give it status “Draft”, and create and check-in the initial draft of the PEP. The PEP editors will not unreasonably deny a PEP. Reasons for denying PEP status include duplication of effort, being technically unsound, not providing proper motivation or addressing backwards compatibility, or not in keeping with the Python philosophy. The BDFL can be consulted during the approval phase, and is the final arbiter of the draft’s PEP-ability.

Developers with hg push privileges for the PEP repository may claim PEP numbers directly by creating and committing a new PEP. When doing so, the developer must handle the tasks that would normally be taken care of by the PEP editors (see PEP Editor Responsibilities & Workflow). This includes ensuring the initial version meets the expected standards for submitting a PEP. Alternately, even developers may choose to submit PEPs through the PEP editors. When doing so, let the PEP editors know you have hg push privileges and they can guide you through the process of updating the PEP repository directly.

As updates are necessary, the PEP author can check in new versions if they (or a collaborating developer) have hg push privileges, or else they can email new PEP versions to the PEP editors for publication.

After a PEP number has been assigned, a draft PEP may be discussed further on python-ideas (getting a PEP number assigned early can be useful for ease of reference, especially when multiple draft PEPs are being considered at the same time). Eventually, all Standards Track PEPs must be sent to the python-dev list for review as described in the next section.

Standards Track PEPs consist of two parts, a design document and a reference implementation. It is generally recommended that at least a prototype implementation be co-developed with the PEP, as ideas that sound good in principle sometimes turn out to be impractical when subjected to the test of implementation.

PEP authors are responsible for collecting community feedback on a PEP before submitting it for review. However, wherever possible, long open-ended discussions on public mailing lists should be avoided. Strategies to keep the discussions efficient include: setting up a separate SIG mailing list for the topic, having the PEP author accept private comments in the early design phases, setting up a wiki page, etc. PEP authors should use their discretion here.
PEP Review & Resolution

Once the authors have completed a PEP, they may request a review for style and consistency from the PEP editors. However, the content and final acceptance of the PEP must be requested of the BDFL, usually via an email to the python-dev mailing list. PEPs are reviewed by the BDFL and his chosen consultants, who may accept or reject a PEP or send it back to the author(s) for revision. For a PEP that is predetermined to be acceptable (e.g., it is an obvious win as-is and/or its implementation has already been checked in) the BDFL may also initiate a PEP review, first notifying the PEP author(s) and giving them a chance to make revisions.

The final authority for PEP approval is the BDFL. However, whenever a new PEP is put forward, any core developer that believes they are suitably experienced to make the final decision on that PEP may offer to serve as the BDFL’s delegate (or “PEP czar”) for that PEP. If their self-nomination is accepted by the other core developers and the BDFL, then they will have the authority to approve (or reject) that PEP. This process happens most frequently with PEPs where the BDFL has granted in principle approval for something to be done, but there are details that need to be worked out before the PEP can be accepted.

If the final decision on a PEP is to be made by a delegate rather than directly by the BDFL, this will be recorded by including the “BDFL-Delegate” header in the PEP.

PEP review and resolution may also occur on a list other than python-dev (for example, distutils-sig for packaging related PEPs that don’t immediately affect the standard library). In this case, the “Discussions-To” heading in the PEP will identify the appropriate alternative list where discussion, review and pronouncement on the PEP will occur.

For a PEP to be accepted it must meet certain minimum criteria. It must be a clear and complete description of the proposed enhancement. The enhancement must represent a net improvement. The proposed implementation, if applicable, must be solid and must not complicate the interpreter unduly. Finally, a proposed enhancement must be “pythonic” in order to be accepted by the BDFL. (However, “pythonic” is an imprecise term; it may be defined as whatever is acceptable to the BDFL. This logic is intentionally circular.) See PEP 2 \(^2\) for standard library module acceptance criteria.

Once a PEP has been accepted, the reference implementation must be completed. When the reference implementation is complete and incorporated into the main source code repository, the status will be changed to “Final”.

A PEP can also be assigned status “Deferred”. The PEP author or an editor can assign the PEP this status when no progress is being made on the PEP. Once a PEP is deferred, a PEP editor can re-assign it to draft status.

A PEP can also be “Rejected”. Perhaps after all is said and done it was not a good idea. It is still important to have a record of this fact. The “Withdrawn” status is similar - it means that the PEP author themselves has decided that the PEP is actually a bad idea, or has accepted that a competing proposal is a better alternative.

When a PEP is Accepted, Rejected or Withdrawn, the PEP should be updated accordingly. In addition to updating the status field, at the very least the Resolution header should be added with a link to the relevant post in the python-dev mailing list archives.

PEPs can also be superseded by a different PEP, rendering the original obsolete. This is intended for Informational PEPs, where version 2 of an API can replace version 1.

\(^{2}\) PEP 2, Procedure for Adding New Modules, Faassen (http://www.python.org/dev/peps/pep-0002)
The possible paths of the status of PEPs are as follows:

Some Informational and Process PEPs may also have a status of “Active” if they are never meant to be completed. E.g. PEP 1 (this PEP).

**PEP Maintenance**

In general, Standards track PEPs are no longer modified after they have reached the Final state. Once a PEP has been completed, the Language and Standard Library References become the formal documentation of the expected behavior.

Informational and Process PEPs may be updated over time to reflect changes to development practices and other details. The precise process followed in these cases will depend on the nature and purpose of the PEP being updated.

**What belongs in a successful PEP?**

Each PEP should have the following parts:

1. **Preamble** – RFC 822 style headers containing meta-data about the PEP, including the PEP number, a short descriptive title (limited to a maximum of 44 characters), the names, and optionally the contact info for each author, etc.

2. **Abstract** – A short (~200 word) description of the technical issue being addressed.

3. **Copyright/public domain** – Each PEP must either be explicitly labeled as placed in the public domain (see this PEP as an example) or licensed under the Open Publication License.

4. **Specification** – The technical specification should describe the syntax and semantics of any new language feature. The specification should be detailed enough to allow competing, interoperable implementations for at least the current major Python platforms (CPython, Jython, IronPython, PyPy).

5. **Motivation** – The motivation is critical for PEPs that want to change the Python language. It should clearly explain why the existing language specification is inadequate to address the problem that the PEP solves. PEP submissions without sufficient motivation may be rejected outright.

6. **Rationale** – The rationale fleshes out the specification by describing what motivated the design and why particular design decisions were made. It should describe alternate designs that were considered and related work, e.g. how the feature is supported in other languages. The rationale should provide evidence of consensus within the community and discuss important objections or concerns raised during discussion.

7. **Backwards Compatibility** – All PEPs that introduce backwards incompatibilities must include a section describing these incompatibilities and their severity. The PEP must explain how the author proposes to deal with these incompatibilities. PEP submissions without a sufficient backwards compatibility treatise may be rejected outright.

8. **Reference Implementation** – The reference implementation must be completed before any PEP is given status “Final”, but it need not be completed before the PEP is accepted. While there is merit to the approach of reaching consensus on the specification and rationale before writing code, the
principle of “rough consensus and running code” is still useful when it comes to resolving many
discussions of API details.

The final implementation must include test code and documentation appropriate for either the Python
language reference or the standard library reference.

**PEP Formats and Templates**

There are two PEP formats available to authors: plaintext and reStructuredText. Both are UTF-8-encoded
text files.

Plaintext PEPs are written with minimal structural markup that adheres to a rigid style. PEP 9 contains a
instructions and a template ³ you can use to get started writing your plaintext PEP.

ReStructuredText PEPs allow for rich markup that is still quite easy to read, but results in much better-
looking and more functional HTML. PEP 12 contains instructions and a template ⁴ for reStructuredText
PEPs.

There is a Python script that converts both styles of PEPs to HTML for viewing on the web ⁵. Parsing
and conversion of plaintext PEPs is self-contained within the script. reStructuredText PEPs are parsed and
converted by Docutils code called from the script.

**PEP Header Preamble**

Each PEP must begin with an RFC 822 style header preamble. The headers must appear in the following
order. Headers marked with “*” are optional and are described below. All other headers are required.

```
PEP: <pep number>
Title: <pep title>
Version: <version string>
Last-Modified: <date string>
Author: <list of authors' real names and optionally, email addrs>
* BDFL-Delegate: <PEP czar's real name>
* Discussions-To: <email address>
Status: <Draft | Active | Accepted | Deferred | Rejected |
        Withdrawn | Final | Superseded>
Type: <Standards Track | Informational | Process>
* Content-Type: <text/plain | text/x-rst>
* Requires: <pep numbers>
Created: <date created on, in dd-mmm-yyyy format>
* Python-Version: <version number>
Post-History: <dates of postings to python-list and python-dev>
* Replaces: <pep number>
* Superseded-By: <pep number>
* Resolution: <url>
```

³ PEP 9, Sample Plaintext PEP Template, Warsaw (http://www.python.org/dev/peps/pep-0009)
⁴ PEP 12, Sample reStructuredText PEP Template, Goodger, Warsaw (http://www.python.org/dev/peps/pep-0012)
⁵ The script referred to here is pep2pyramid.py, the successor to pep2html.py, both of which live in the same directory in
the hg repo as the PEPs themselves. Try pep2html.py --help for details. The URL for viewing PEPs on the web is
http://www.python.org/dev/peps/.
The Author header lists the names, and optionally the email addresses of all the authors/owners of the PEP. The format of the Author header value must be

Random J. User <address@dom.ain>

if the email address is included, and just

Random J. User

if the address is not given. For historical reasons the format “address@dom.ain (Random J. User)” may appear in a PEP, however new PEPs must use the mandated format above, and it is acceptable to change to this format when PEPs are updated.

If there are multiple authors, each should be on a separate line following RFC 2822 continuation line conventions. Note that personal email addresses in PEPs will be obscured as a defense against spam harvesters.

The BDFL-Delegate field is used to record cases where the final decision to approve or reject a PEP rests with someone other than the BDFL. (The delegate’s email address is currently omitted due to a limitation in the email address masking for reStructuredText PEPs)

Note: The Resolution header is required for Standards Track PEPs only. It contains a URL that should point to an email message or other web resource where the pronouncement about the PEP is made.

For a PEP where final pronouncement will be made on a list other than python-dev, a Discussions-To header will indicate the mailing list or URL where the pronouncement will occur. A temporary Discussions-To header may also be used when a draft PEP is being discussed prior to submission for pronouncement. No Discussions-To header is necessary if the PEP is being discussed privately with the author, or on the python-list, python-ideas or python-dev mailing lists. Note that email addresses in the Discussions-To header will not be obscured.

The Type header specifies the type of PEP: Standards Track, Informational, or Process.

The format of a PEP is specified with a Content-Type header. The acceptable values are “text/plain” for plaintext PEPs (see PEP 9) and “text/x-rst” for reStructuredText PEPs (see PEP 12). Plaintext (“text/plain”) is the default if no Content-Type header is present.

The Created header records the date that the PEP was assigned a number, while Post-History is used to record the dates of when new versions of the PEP are posted to python-list and/or python-dev. Both headers should be in dd-mmm-yyyy format, e.g. 14-Aug-2001.

Standards Track PEPs will typically have a Python-Version header which indicates the version of Python that the feature will be released with. Standards Track PEPs without a Python-Version header indicate interoperability standards that will initially be supported through external libraries and tools, and then supplemented by a later PEP to add support to the standard library. Informational and Process PEPs do not need a Python-Version header.

PEPs may have a Requires header, indicating the PEP numbers that this PEP depends on.

PEPs may also have a Superseded-By header indicating that a PEP has been rendered obsolete by a later document; the value is the number of the PEP that replaces the current document. The newer PEP must have a Replaces header containing the number of the PEP that it rendered obsolete.

4.2. Knowledge Base Articles 41
Auxiliary Files

PEPs may include auxiliary files such as diagrams. Such files must be named pep-XXXX-Y.ext, where “XXXX” is the PEP number, “Y” is a serial number (starting at 1), and “ext” is replaced by the actual file extension (e.g. “png”).

Reporting PEP Bugs, or Submitting PEP Updates

How you report a bug, or submit a PEP update depends on several factors, such as the maturity of the PEP, the preferences of the PEP author, and the nature of your comments. For the early draft stages of the PEP, it’s probably best to send your comments and changes directly to the PEP author. For more mature, or finished PEPs you may want to submit corrections to the Python issue tracker so that your changes don’t get lost. If the PEP author is a Python developer, assign the bug/patch to them, otherwise assign it to a PEP editor.

When in doubt about where to send your changes, please check first with the PEP author and/or a PEP editor. PEP authors with hg push privileges for the PEP repository can update the PEPs themselves by using “hg push” to submit their changes.

Transferring PEP Ownership

It occasionally becomes necessary to transfer ownership of PEPs to a new champion. In general, it is preferable to retain the original author as a co-author of the transferred PEP, but that’s really up to the original author. A good reason to transfer ownership is because the original author no longer has the time or interest in updating it or following through with the PEP process, or has fallen off the face of the ‘net (i.e. is unreachable or not responding to email). A bad reason to transfer ownership is because the author doesn’t agree with the direction of the PEP. One aim of the PEP process is to try to build consensus around a PEP, but if that’s not possible, an author can always submit a competing PEP.

If you are interested in assuming ownership of a PEP, send a message asking to take over, addressed to both the original author and the PEP editors <peps@python.org>. If the original author doesn’t respond to email in a timely manner, the PEP editors will make a unilateral decision (it’s not like such decisions can’t be reversed :).

PEP Editor Responsibilities & Workflow

A PEP editor must subscribe to the <peps@python.org> list. All correspondence related to PEP administration should be sent (or forwarded) to <peps@python.org> (but please do not cross-post!).

For each new PEP that comes in an editor does the following:

- Read the PEP to check if it is ready: sound and complete. The ideas must make technical sense, even if they don’t seem likely to be accepted.
- The title should accurately describe the content.
- Edit the PEP for language (spelling, grammar, sentence structure, etc.), markup (for reST PEPs), code style (examples should match PEP 8 & 7).
If the PEP isn’t ready, an editor will send it back to the author for revision, with specific instructions.

Once the PEP is ready for the repository, a PEP editor will:

- Assign a PEP number (almost always just the next available number, but sometimes it’s a special/joke number, like 666 or 3141). (Clarification: For Python 3, numbers in the 3000s were used for Py3k-specific proposals. But now that all new features go into Python 3 only, the process is back to using numbers in the 100s again. Remember that numbers below 100 are meta-PEPs.)

- Add the PEP to a local clone of the PEP repository. For mercurial workflow instructions, follow The Python Developers Guide

The mercurial repo for the peps is:

http://hg.python.org/peps/

- Run .genpepindex.py and ./pep2html.py <PEP Number> to ensure they are generated without errors. If either triggers errors, then the web site will not be updated to reflect the PEP changes.

- Commit and push the new (or updated) PEP

- Monitor python.org to make sure the PEP gets added to the site properly. If it fails to appear, running make will build all of the current PEPs. If any of these are triggering errors, they must be corrected before any PEP will update on the site.

- Send email back to the PEP author with next steps (post to python-list & -dev).

Updates to existing PEPs also come in to peps@python.org. Many PEP authors are not Python committers yet, so PEP editors do the commits for them.

Many PEPs are written and maintained by developers with write access to the Python codebase. The PEP editors monitor the python-checkins list for PEP changes, and correct any structure, grammar, spelling, or markup mistakes they see.

PEP editors don’t pass judgment on PEPs. They merely do the administrative & editorial part (which is generally a low volume task).

Resources:

- Index of Python Enhancement Proposals

- Following Python’s Development

- Python Developer’s Guide

- Frequently Asked Questions for Developers

References and Footnotes

Copyright

This document has been placed in the public domain.
4.2.3 Sample reStructuredText PEP Template

PEP 12
Title Sample reStructuredText PEP Template
Version $Revision$
Last-Modified $Date$
Author David Goodger <goodger@python.org>, Barry Warsaw <barry@python.org>
Status Active
Type Process
Content-Type text/x-rst
Created 05-Aug-2002
Post-History 30-Aug-2002

Abstract

This PEP provides a boilerplate or sample template for creating your own reStructuredText PEPs. In conjunction with the content guidelines in PEP 1 \(^1\), this should make it easy for you to conform your own PEPs to the format outlined below.

Note: if you are reading this PEP via the web, you should first grab the text (reStructuredText) source of this PEP in order to complete the steps below. **DO NOT USE THE HTML FILE AS YOUR TEMPLATE!**

The source for this (or any) PEP can be found in the PEPs repository, viewable on the web at https://hg.python.org/peps/file/tip.

If you would prefer not to use markup in your PEP, please see PEP 9, “Sample Plaintext PEP Template” \(^2\).

Rationale

PEP submissions come in a wide variety of forms, not all adhering to the format guidelines set forth below. Use this template, in conjunction with the format guidelines below, to ensure that your PEP submission won’t get automatically rejected because of form.

ReStructuredText is offered as an alternative to plaintext PEPs, to allow PEP authors more functionality and expressivity, while maintaining easy readability in the source text. The processed HTML form makes the functionality accessible to readers: live hyperlinks, styled text, tables, images, and automatic tables of contents, among other advantages. For an example of a PEP marked up with reStructuredText, see PEP 287.

\(^1\) PEP 1, PEP Purpose and Guidelines, Warsaw, Hylton (http://www.python.org/dev/peps/pep-0001)
\(^2\) PEP 9, Sample Plaintext PEP Template, Warsaw (http://www.python.org/dev/peps/pep-0009)
How to Use This Template

To use this template you must first decide whether your PEP is going to be an Informational or Standards Track PEP. Most PEPs are Standards Track because they propose a new feature for the Python language or standard library. When in doubt, read PEP 1 for details or contact the PEP editors <peps@python.org>.

Once you’ve decided which type of PEP yours is going to be, follow the directions below.

• Make a copy of this file (.txt file, not HTML!) and perform the following edits.
• Replace the “PEP: 12” header with “PEP: XXX” since you don’t yet have a PEP number assignment.
• Change the Title header to the title of your PEP.
• Leave the Version and Last-Modified headers alone; we’ll take care of those when we check your PEP into Python’s Subversion repository. These headers consist of keywords (“Revision” and “Date” enclosed in “$”-signs) which are automatically expanded by the repository. Please do not edit the expanded date or revision text.
• Change the Author header to include your name, and optionally your email address. Be sure to follow the format carefully: your name must appear first, and it must not be contained in parentheses. Your email address may appear second (or it can be omitted) and if it appears, it must appear in angle brackets. It is okay to obfuscate your email address.
• If there is a mailing list for discussion of your new feature, add a Discussions-To header right after the Author header. You should not add a Discussions-To header if the mailing list to be used is either python-list@python.org or python-dev@python.org, or if discussions should be sent to you directly. Most Informational PEPs don’t have a Discussions-To header.
• Change the Status header to “Draft”.
• For Standards Track PEPs, change the Type header to “Standards Track”.
• For Informational PEPs, change the Type header to “Informational”.
• For Standards Track PEPs, if your feature depends on the acceptance of some other currently in-development PEP, add a Requires header right after the Type header. The value should be the PEP number of the PEP yours depends on. Don’t add this header if your dependent feature is described in a Final PEP.
• Change the Created header to today’s date. Be sure to follow the format carefully: it must be in dd–mmm–yyyy format, where the mmm is the 3 English letter month abbreviation, i.e. one of Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec.
• For Standards Track PEPs, after the Created header, add a Python-Version header and set the value to the next planned version of Python, i.e. the one your new feature will hopefully make its first appearance in. Do not use an alpha or beta release designation here. Thus, if the last version of Python was 2.2 alpha 1 and you’re hoping to get your new feature into Python 2.2, set the header to:

    Python-Version: 2.2

• Leave Post-History alone for now; you’ll add dates to this header each time you post your PEP to python-list@python.org or python-dev@python.org. If you posted your PEP to the lists on August 14, 2001 and September 3, 2001, the Post-History header would look like:
You must manually add new dates and check them in. If you don’t have check-in privileges, send your changes to the PEP editors.

- Add a Replaces header if your PEP obsoletes an earlier PEP. The value of this header is the number of the PEP that your new PEP is replacing. Only add this header if the older PEP is in “final” form, i.e. is either Accepted, Final, or Rejected. You aren’t replacing an older open PEP if you’re submitting a competing idea.

- Now write your Abstract, Rationale, and other content for your PEP, replacing all this gobbledygook with your own text. Be sure to adhere to the format guidelines below, specifically on the prohibition of tab characters and the indentation requirements.

- Update your References and Copyright section. Usually you’ll place your PEP into the public domain, in which case just leave the Copyright section alone. Alternatively, you can use the Open Publication License, but public domain is still strongly preferred.

- Leave the Emacs stanza at the end of this file alone, including the formfeed character (“^L”, or \f).

- Send your PEP submission to the PEP editors at peps@python.org.

ReStructuredText PEP Formatting Requirements

The following is a PEP-specific summary of reStructuredText syntax. For the sake of simplicity and brevity, much detail is omitted. For more detail, see Resources below. Literal blocks (in which no markup processing is done) are used for examples throughout, to illustrate the plaintext markup.

General

You must adhere to the Emacs convention of adding two spaces at the end of every sentence. You should fill your paragraphs to column 70, but under no circumstances should your lines extend past column 79. If your code samples spill over column 79, you should rewrite them.

Tab characters must never appear in the document at all. A PEP should include the standard Emacs stanza included by example at the bottom of this PEP.

Section Headings

PEP headings must begin in column zero and the initial letter of each word must be capitalized as in book titles. Acronyms should be in all capitals. Section titles must be adorned with an underline, a single repeated punctuation character, which begins in column zero and must extend at least as far as the right edge of the title text (4 characters minimum). First-level section titles are underlined with “=” (equals signs), second-level section titles with “-” (hyphens), and third-level section titles with “’” (single quotes or apostrophes). For example:

First-Level Title
================
If there are more than three levels of sections in your PEP, you may insert overline/underline-adorned titles for the first and second levels as follows:

============================
First-Level Title (optional)
============================

----------------------------
Second-Level Title (optional)
----------------------------

Third-Level Title

Fourth-Level Title

Fifth-Level Title

You shouldn’t have more than five levels of sections in your PEP. If you do, you should consider rewriting it.

You must use two blank lines between the last line of a section’s body and the next section heading. If a subsection heading immediately follows a section heading, a single blank line in-between is sufficient.

The body of each section is not normally indented, although some constructs do use indentation, as described below. Blank lines are used to separate constructs.

**Paragraphs**

Paragraphs are left-aligned text blocks separated by blank lines. Paragraphs are not indented unless they are part of an indented construct (such as a block quote or a list item).

**Inline Markup**

Portions of text within paragraphs and other text blocks may be styled. For example:

Text may be marked as *emphasized* (single asterisk markup, typically shown in italics) or **strongly emphasized** (double asterisks, typically boldface). ```Inline literals``` (using double backquotes) are typically rendered in a monospaced typeface. No further markup recognition is done within the double backquotes, so they're safe for any kind of code snippets.
Block Quotes

Block quotes consist of indented body elements. For example:

This is a paragraph.

    This is a block quote.

    A block quote may contain many paragraphs.

Block quotes are used to quote extended passages from other sources. Block quotes may be nested inside other body elements. Use 4 spaces per indent level.

Literal Blocks

Literal blocks are used for code samples or preformatted ASCII art. To indicate a literal block, preface the indented text block with “::” (two colons). The literal block continues until the end of the indentation. Indent the text block by 4 spaces. For example:

This is a typical paragraph. A literal block follows.

    ::

        for a in [5,4,3,2,1]: # this is program code, shown as-is
            print a
        print "it's..."
        # a literal block continues until the indentation ends

The paragraph containing only “::” will be completely removed from the output; no empty paragraph will remain. “::” is also recognized at the end of any paragraph. If immediately preceded by whitespace, both colons will be removed from the output. When text immediately precedes the “::”, one colon will be removed from the output, leaving only one colon visible (i.e., “::” will be replaced by “:”). For example, one colon will remain visible here:

Paragraph::

    Literal block

Lists

Bullet list items begin with one of “-”, “*”, or “+” (hyphen, asterisk, or plus sign), followed by whitespace and the list item body. List item bodies must be left-aligned and indented relative to the bullet; the text immediately after the bullet determines the indentation. For example:

This paragraph is followed by a list.

* This is the first bullet list item. The blank line above the first list item is required; blank lines between list items (such as below this paragraph) are optional.
* This is the first paragraph in the second item in the list.

This is the second paragraph in the second item in the list.
The blank line above this paragraph is required. The left edge
of this paragraph lines up with the paragraph above, both
indented relative to the bullet.

- This is a sublist. The bullet lines up with the left edge of
  the text blocks above. A sublist is a new list so requires a
  blank line above and below.

* This is the third item of the main list.

This paragraph is not part of the list.

Enumerated (numbered) list items are similar, but use an enumerator instead of a bullet. Enumerators are
numbers (1, 2, 3, ...), letters (A, B, C, ...; uppercase or lowercase), or Roman numerals (i, ii, iii, iv, ...;
uppercase or lowercase), formatted with a period suffix (“1.”, “2.”), parentheses (“(1)”, “(2)”), or a right-
parenthesis suffix (“1)”, “2)”). For example:

1. As with bullet list items, the left edge of paragraphs must
   align.

2. Each list item may contain multiple paragraphs, sublists, etc.

   This is the second paragraph of the second list item.

   a) Enumerated lists may be nested.
   b) Blank lines may be omitted between list items.

Definition lists are written like this:

what

Definition lists associate a term with a definition.

how

The term is a one-line phrase, and the definition is one
or more paragraphs or body elements, indented relative to
the term.

Tables

Simple tables are easy and compact:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A and B</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
There must be at least two columns in a table (to differentiate from section titles). Column spans use underlines of hyphens (“Inputs” spans the first two columns):

```
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
```

Text in a first-column cell starts a new row. No text in the first column indicates a continuation line; the rest of the cells may consist of multiple lines. For example:

```
<table>
<thead>
<tr>
<th>col 1</th>
<th>col 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Second column of row 1.</td>
</tr>
<tr>
<td>2</td>
<td>Second column of row 2.</td>
</tr>
<tr>
<td>3</td>
<td>Second line of paragraph.</td>
</tr>
<tr>
<td></td>
<td>- Second column of row 3.</td>
</tr>
<tr>
<td></td>
<td>- Second item in bullet</td>
</tr>
<tr>
<td></td>
<td>list (row 3, column 2).</td>
</tr>
</tbody>
</table>
```

Hyperlinks

When referencing an external web page in the body of a PEP, you should include the title of the page in the text, with either an inline hyperlink reference to the URL or a footnote reference (see Footnotes below). Do not include the URL in the body text of the PEP.

Hyperlink references use backquotes and a trailing underscore to mark up the reference text; backquotes are optional if the reference text is a single word. For example:

```
In this paragraph, we refer to the `Python web site`_.
```

An explicit target provides the URL. Put targets in a References section at the end of the PEP, or immediately after the reference. Hyperlink targets begin with two periods and a space (the “explicit markup start”), followed by a leading underscore, the reference text, a colon, and the URL (absolute or relative):

```
.. _Python web site: http://www.python.org/
```

The reference text and the target text must match (although the match is case-insensitive and ignores differences in whitespace). Note that the underscore trails the reference text but precedes the target text. If you think of the underscore as a right-pointing arrow, it points away from the reference and toward the target.

The same mechanism can be used for internal references. Every unique section title implicitly defines an internal hyperlink target. We can make a link to the Abstract section like this:
Here is a hyperlink reference to the `Abstract`_ section. The backquotes are optional since the reference text is a single word; we can also just write: Abstract_.

Footnotes containing the URLs from external targets will be generated automatically at the end of the References section of the PEP, along with footnote references linking the reference text to the footnotes.

Text of the form “PEP x” or “RFC x” (where “x” is a number) will be linked automatically to the appropriate URLs.

**Footnotes**

Footnote references consist of a left square bracket, a number, a right square bracket, and a trailing underscore:

This sentence ends with a footnote reference [1]_.

Whitespace must precede the footnote reference. Leave a space between the footnote reference and the preceding word.

When referring to another PEP, include the PEP number in the body text, such as “PEP 1”. The title may optionally appear. Add a footnote reference following the title. For example:

Refer to PEP 1 [2]_ for more information.

Add a footnote that includes the PEP’s title and author. It may optionally include the explicit URL on a separate line, but only in the References section. Footnotes begin with “.. ” (the explicit markup start), followed by the footnote marker (no underscores), followed by the footnote body. For example:

References
==========

If you decide to provide an explicit URL for a PEP, please use this as the URL template:

http://www.python.org/dev/peps/pep-xxxx

PEP numbers in URLs must be padded with zeros from the left, so as to be exactly 4 characters wide, however PEP numbers in the text are never padded.

During the course of developing your PEP, you may have to add, remove, and rearrange footnote references, possibly resulting in mismatched references, obsolete footnotes, and confusion. Auto-numbered footnotes allow more freedom. Instead of a number, use a label of the form “#word”, where “word” is a mnemonic consisting of alphanumerics plus internal hyphens, underscores, and periods (no whitespace or other characters are allowed). For example:

Refer to PEP 1 [#PEP-1]_ for more information.

References
==========

4.2. Knowledge Base Articles
Footnotes and footnote references will be numbered automatically, and the numbers will always match. Once a PEP is finalized, auto-numbered labels should be replaced by numbers for simplicity.

Images

If your PEP contains a diagram, you may include it in the processed output using the “image” directive:

```
.. image:: diagram.png
```

Any browser-friendly graphics format is possible: .png, .jpeg, .gif, .tiff, etc.

Since this image will not be visible to readers of the PEP in source text form, you should consider including a description or ASCII art alternative, using a comment (below).

Comments

A comment block is an indented block of arbitrary text immediately following an explicit markup start: two periods and whitespace. Leave the “..” on a line by itself to ensure that the comment is not misinterpreted as another explicit markup construct. Comments are not visible in the processed document. For the benefit of those reading your PEP in source form, please consider including a description of or ASCII art alternatives to any images you include. For example:

```
.. image:: dataflow.png

.. Data flows from the input module, through the "black box" module, and finally into (and through) the output module.
```

The Emacs stanza at the bottom of this document is inside a comment.

Escaping Mechanism

reStructuredText uses backslashes (“\”) to override the special meaning given to markup characters and get the literal characters themselves. To get a literal backslash, use an escaped backslash (“\\”). There are two contexts in which backslashes have no special meaning: literal blocks and inline literals (see Inline Markup above). In these contexts, no markup recognition is done, and a single backslash represents a literal backslash, without having to double up.

If you find that you need to use a backslash in your text, consider using inline literals or a literal block instead.
Habits to Avoid

Many programmers who are familiar with TeX often write quotation marks like this:

```
single-quoted' or ``double-quoted''
```

Backquotes are significant in reStructuredText, so this practice should be avoided. For ordinary text, use ordinary 'single-quotes' or “double-quotes”. For inline literal text (see Inline Markup above), use double-backquotes:

```
``literal text: in here, anything goes!``
```

Resources

Many other constructs and variations are possible. For more details about the reStructuredText markup, in increasing order of thoroughness, please see:

- A ReStructuredText Primer, a gentle introduction.
- Quick reStructuredText, a users’ quick reference.
- reStructuredText Markup Specification, the final authority.

The processing of reStructuredText PEPs is done using Docutils. If you have a question or require assistance with reStructuredText or Docutils, please post a message to the Docutils-users mailing list. The Docutils project web site has more information.

References

Copyright

This document has been placed in the public domain.
Shared Learning

Shared learning is the process of working collectively to achieve a common objective in a group. Team members tend to share knowledge and complement each others’ skills. If there is no commitment and effort from team members then working and learning from team work may fail.

Main principles

- Creating the context and environment for empowered action
- Learning from experience - what has and has not worked well
- A multi-disciplinary approach to addressing complex issues
- Acknowledging that people on the team have the capability and capacity to effect change
- Utilizing the resources on the team fully

Index of shared learning topics

Section author: Le Yang <le.yang@leidoshealth.com>

5.1 Automation of Repetitive Tasks

“The computer is incredibly fast, accurate, and stupid. Man is unbelievably slow, inaccurate, and brilliant. The marriage of the two is a force beyond calculation.”

— Leo Cherne

5.1.1 Philosophy

While contents are likely different with each change, much of our work are of repetitive nature at the process level. Instead of repeating the same process every time with different content, as long the process can be defined discretely at each step with deterministic outcomes, it can be automated. This approach has the two major advantages:

Improved Quality and Reliability Automation is precise and repeatable. It ensures the final product or outcome is produced following the same specifications and process every time.
**Improved Productivity**  With overwhelmingly more computational power, automation allows tasks to be completed in reduced time, thus improved productivity.

### 5.1.2 Tools

#### AWK

The AWK language is a data-driven scripting language consisting of a set of actions to be taken against streams of textual data – either run directly on files or used as part of a pipeline – for purposes of extracting or transforming text, such as producing formatted reports. The language extensively uses the string datatype, associative arrays (that is, arrays indexed by key strings), and regular expressions. While AWK has a limited intended application domain and was especially designed to support one-liner programs.

One of the authors described AWK as "a language for processing text files. A file is treated as a sequence of records, and by default each line is a record. Each line is broken up into a sequence of fields, so we can think of the first word in a line as the first field, the second word as the second field, and so on. An AWK program is of a sequence of pattern-action statements. AWK reads the input a line at a time. A line is scanned for each pattern in the program, and for each pattern that matches, the associated action is executed."

#### Sample CSV file

<table>
<thead>
<tr>
<th>CostCenter, CDM, Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7021, 70211234, &quot;a charge&quot;</td>
</tr>
<tr>
<td>7023, 70231234, &quot;another charge&quot;</td>
</tr>
<tr>
<td>7021, 70215678, &quot;a charge&quot;</td>
</tr>
</tbody>
</table>

#### Python

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

Highlights of Python features:

- Simple
- Easy to learn
- Free and open source
- High-level language
- Portable
- Interpreted
- Object oriented
- Extensible
• Embeddable
• Extensive libraries

See A Byte of Python for introduction for beginners.

**Structured Query Language (SQL) and ETL (Extract, Transform, Load) processes**

By extracting data from the SCM Repository directly, SQL queries can be used to produce data formatted fitting to the requirement that would normally take much time longer to perform manually.

### 5.1.3 Example: Order Availability Policy (OAP) creation

When there is a need to create new OAP, or update existing OAPs, the combination of SQL and Express Load is used: SQL query to produce the load content, and Express Load to perform the addition or update. Below is a high-level overview of that process:

- Create model according to *OAP model: facility, abbreviation, LFG, and location prefix.*
- Produce OAP parent definition from OAP name

<table>
<thead>
<tr>
<th>UniqueRef</th>
<th>Name</th>
<th>(No column name)</th>
<th>(No column name)</th>
<th>(No column name)</th>
<th>(No column name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>All Hidden_CXE</td>
<td>TRUE</td>
<td>Parent Order 1</td>
<td>FALSE</td>
<td>Not Available</td>
</tr>
<tr>
<td>2</td>
<td>CFJKPRTWX_ADEM</td>
<td>TRUE</td>
<td>Parent Order 1</td>
<td>FALSE</td>
<td>Not Available</td>
</tr>
</tbody>
</table>

- Cross apply model by OAP name
- If applicable, deactivate existing OAP detail definition entries

**Important:**

- This is needed to prevent duplication of OAP detail definition entries when an existing OAP is updated
- The deactivation entries must proceed any new definition entries (below)

- Apply dither rule following *OAP nomenclature and definition (non-Rx)* to produce OAP detail definition
• Copy OAP parent and detail definitions to blank Workbook (Catalog)

• Export Workbook to load files (.blk)

• Express load .blk files to add/update desired OAP

See also:

Configuration documentation on *Order Availability Policy*.

### 5.1.4 Example: Batch build of charge orders and order sets

This section pertain only to the build for ministries using Soarian Financials. Soarian Financials interfaces with SCM as a single system spanning multiple ministries, meaning it uses the same *CDM* for all captured charges. This nature makes building charge orders and order sets a process that can be well defined, making it a great candidate for automation.

Soarian Financials uses *SP* to designate cost centers, and *SID* to designate charge items. *SPSID* is then the unique identifier for a charge item for a cost center.
Building charge orders

1. Update source CSV file with CDMs: SPSID, CDM description, and charge prefix are defined in the CSV file. This is the most time consuming part of this process. However with the simple data structure the source CSV file can be produced much easier, almost directly, after analyzing the daily CDM update notifications.

<table>
<thead>
<tr>
<th>CDM</th>
<th>Name</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>742034213</td>
<td>Negative Pressure Wound Therapy Less Than 50 cm</td>
<td>WHC</td>
</tr>
<tr>
<td>742034214</td>
<td>Negative Pressure Wound Therapy Greater Than 50 cm</td>
<td>WHC</td>
</tr>
</tbody>
</table>

2. A Python script runs to apply the charge order template to each data record in the source CSV file. Substitutions happen only for variations, e.g. charge name and description, charge code, charge build date, etc. according to Soarian Charge Build Standard. Click charge order template to view the template.

3. The outcome of the script is a .blk file suitable to express load.

4. Update special charge orders (e.g. timed charges) with correct order entry form.

5. Express load the .blk file and profit.

Building charge order sets

Batch building charge order sets follow similar steps, except different template files are used (see templates for charge order set properties and items to include in charge order set and mapping). There is of course the pre-requisite that charge orders must exist in the catalog or error will occur during express load. Below is the source table used to assign charge orders to new/updated charge order sets:

<table>
<thead>
<tr>
<th>Charge order name</th>
<th>Multi order field</th>
<th>Multi order field index</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHG NUR Intra-Op Neuro Testing Per 15 Min 713036849</td>
<td>Multi-OrderCheckbox</td>
<td>2</td>
</tr>
<tr>
<td>CHG NUR MEP Upper + Lower Limbs 713033201</td>
<td>Multi-OrderCheckbox</td>
<td>2</td>
</tr>
<tr>
<td>CHG NUR Needle EMG 2 Extremities w/wo Related Paraspinal Areas 713010164</td>
<td>MultiOrderGrid</td>
<td>9</td>
</tr>
<tr>
<td>CHG NUR Needle EMG Cranial Nerve Bilateral 713033887</td>
<td>MultiOrderGrid</td>
<td>9</td>
</tr>
<tr>
<td>CHG NUR Needle EMG Cranial Nerve Unilateral 713033217</td>
<td>MultiOrderGrid</td>
<td>9</td>
</tr>
<tr>
<td>CHG NUR SEP Nerves Lower Limbs 713030192</td>
<td>MultiOrderGrid</td>
<td>9</td>
</tr>
<tr>
<td>CHG NUR SEP Nerves Upper + Lower Limbs 713033200</td>
<td>MultiOrderGrid</td>
<td>8</td>
</tr>
<tr>
<td>CHG NUR SEP Nerves Upper Limbs 713030191</td>
<td>MultiOrderGrid</td>
<td>8</td>
</tr>
</tbody>
</table>

5.1. Automation of Repetitive Tasks
5.2 An Introduction to MLM

5.2.1 What is an MLM?

A Medical Logic Module (MLM) is an independent unit in a health knowledge base. Each MLM contains maintenance information, links to other sources of knowledge, and enough logic to make a single health decision. MLMs are called by a program - an event monitor - when the condition they are written to help with occurs.

Medical logic modules are written in Arden Syntax, a language used for representing and sharing medical knowledge. It is used to generate alerts, interpretations, screen and manage messages. Clinical and scientific knowledge is represented by using this extensively recognized standard in an executable format which can be used by Clinical decision support systems. A Vital task of syntax is to share medical knowledge base across many institutions. Arden syntax was formerly a standard under ASTM, which was published in the year 1992 and is now part of Health Level Seven International. Arden syntax version 2.0 was published by HL7 in 1999, which is responsible for developing all the latest versions. Arden syntax version 2.9 is the current version.

Arden Syntax is both a specification for MLM format, and a set of programming language grammer for use in MLMs.

5.2.2 MLM structure and grammar

Document structure

An MLM begins with the marker `maintenance:` and ends with the marker `end:`. An MLM is composed of slots grouped into three required categories: `maintenance`, `library`, and `knowledge`, and one optional category, `resources`. A category is indicated by a category name followed immediately by a colon (that is, `maintenance:`, `library:`, `knowledge:`, and `resources:`).

Within each category is a set of slots. Each slot consists of a slot name, followed immediately by a colon (for example, `title:`), then followed by the slot body, and terminated with two adjacent semicolons (`;;`) which is referred to as double semicolon. Each slot must be unique in the MLM, and categories and slots must follow the order in which they are listed in the standard. Some slots are required and others are optional.

The end of the MLM is marked by the word `end` followed immediately by a colon (that is, `end:`). White space may precede the terminator and follow the colon but no white space is allowed between the terminator and the colon.

Arden Syntax grammar

- Statements are basic constructs of an MLM. Each statement specifies a logical constraint or an action to be performed. Statements are terminated with a semicolon (`;`).
• A comment is a string of characters that begins with /* and ends with */. A comment may also be specified by the characters // through line break.

• Any string of spaces, carriage returns, line feeds, horizontal tabs, vertical tabs, form feeds, and comments is known as white space. White space is required between any two tokens that may begin or end with letters, digits, or underscores (for example, if done). They are also required between two string constants.

• The syntax is insensitive to the case of identifiers.

• Identifiers are alphanumeric tokens. The first character of an identifier must be a letter, and the rest must be letters, digits, and underscores (_). Identifiers must be 1 to 80 characters in length. Identifiers are used to represent variables, which hold data.

• Reserved words are predefined tokens made of letters and digits used to construct statements, to represent operators, and to represent data constants. Reserved words cannot be used as identifiers.

• Statements are composed of reserved words, special symbols, and expressions. Expressions may contain any of the following:

  – Constant: The data value may be represented explicitly using a constant like the number 3, the time 1991-03-23T00:00:00, etc.

  – Variable: An identifier within an expression signifies a variable.

  – Operator and Arguments: An expression may contain an operator and one or more sub-expressions known as arguments.

### Samples

Below is a sample blank MLM:

```mlm
maintenance:
  title: ;;
  mlmname: put_mlm_name_here;;
  arden: version 2.5;;
  version: 0.00;;
  institution: ;;
  author: ;;
  specialist: ;;
  date: YYYY-MM-DD;;
  validation: testing;;

library:
  purpose:
  ;;
  explanation:
  ;;
  keywords:
  ;;
  citations:
  ;;
  knowledge:
    type: data-driven;;
```

5.2. An Introduction to MLM
Or, view another example of a more developed MLM.

**MLM execution cycle**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evoke</td>
<td>Events to trigger the MLM.</td>
</tr>
<tr>
<td>Data</td>
<td>Retrieve data for processing from evoked events or other sources.</td>
</tr>
<tr>
<td>Logic</td>
<td>Process clinical or business rules to conclude true or false.</td>
</tr>
<tr>
<td>Action</td>
<td>Code block to perform action. Only executed if concluded true from the logic slot.</td>
</tr>
</tbody>
</table>

5.2.3 Allscripts Clinical Decision Support

**MLM Editor and CDS**

Use the MLM Editor, an Allscripts application, to customize, create, or view the version history of MLMs. MLMs contain both the instructions for the behavior of alerts and the text that appears in alerts. The Clinical Decision Support (CDS) system provides the ability to create clinically relevant alerts about the patient at the point of care.

Users need *Can maintain medical logic modules* system-supplied right to use the MLM Editor.
MLM development process

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New/Import/Load MLM</td>
<td>Start anew or load existing MLM for editing.</td>
</tr>
<tr>
<td>Repository</td>
<td>Primary Repository of patient data</td>
</tr>
<tr>
<td>Code MLM content</td>
<td>While not required, Primary Repository is often queried to obtain additional data.</td>
</tr>
<tr>
<td>Syntax check</td>
<td>Editor will halt if syntax grammar error is detected.</td>
</tr>
<tr>
<td>Select test case</td>
<td>Select a test visit as the use case to perform the execution test. During execution test, MLM is run within the context of the use case for data retrieval and processing.</td>
</tr>
<tr>
<td>Execution test</td>
<td>Editor will halt if run-time error is detected.</td>
</tr>
<tr>
<td>Save with Beta/Production status</td>
<td>Beta status allows an MLM to work with limited exposures, requiring additional security right to interact with the MLM. Production status makes the MLM accessible to all users who can interact with CDS alerts.</td>
</tr>
<tr>
<td>Restart CDS service</td>
<td>Required for MLMs evoked with time-delayed events.</td>
</tr>
</tbody>
</table>

5.2.4 Further reading

- Arden Syntax v2.7
- SCM Clinical Decision Support MLM Authoring Reference Guide
- SCM Clinical Decision Support Standard MLM Configuration Guide
- SCM Clinical Decision Support Sample MLM Reference Guide

5.3 HL7 (SCM) Specification Development

5.3.1 Introduction

The goal of this article is to cover common process and steps involved in developing HL7 interface specifications for SCM.
Note: Interfaces not communicated using HL7 (e.g. the LMR extract) are outside the scope of this document.

The *SCM HL7 Mapping Guide* is the definitive reference for SCM specific structures.

**SCM interface diagram**

Below is a very generic diagram of how SCM interface processes and interacts with third-party systems:
5.3.2 Interface specification development

Activity diagram

<table>
<thead>
<tr>
<th>Business Analyst</th>
<th>Interface Analyst</th>
<th>Interface Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gather requirements</td>
<td>Review ancillary interface</td>
<td>Interface implementation</td>
</tr>
<tr>
<td></td>
<td>Feasibility research</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Finalize requirements</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Create use cases</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Create mapping specification</td>
<td></td>
</tr>
<tr>
<td>Validate interface</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Explanation of activities

Business Analyst

- Gather requirements
  - What business or clinical objectives will the interface accomplish?
  - What are the required data exchange elements? Where can all relevant data be validated?
  - Define functional requirements - what the interface is supposed to do
Interface Analyst

| Review ancillary interface | • Identify required data elements or processing mode  
|                           | • Understand capability and features of ancillary interface  
|                           | • Document differences between SCM and ancillary interfaces for review with the Business Analyst. |
| Feasibility research      | • Identify business requirements not supported by SCM interface, the ancillary interface, or both interfaces  
|                           | • Provide feedbacks to Business Analyst |
| Create interface mapping specification | • Define non-functional requirements - what the interface is supposed to be. These includes, but not limited to, system connectivity, firewall/routing, port used, etc.  
|                           | • Document any required translation between interfaced systems. |

Interface Developer

| Interface implementation | • Implement/code interface per specification provided by Interface Analyst.  
|                         | • Make revision to interface coding as needed. |

Section author: Le Yang <le.yang@leidoshealth.com>

5.4 Separation of Presentation and Content

Separation of presentation and content is a common idiom, a design philosophy, and a methodology applied in the context of various publishing technology disciplines, including information retrieval, template processing, web design, web development, word processing, desktop publishing, and model-driven development.
5.4.1 Concept

Intended usage

When invoked as an idiom, the underlying concept is to make a distinction between the actual meaning of a document, and how this meaning is presented to its readers. A common example is the `<em>`...`</em>` phrase element in HTML, to denote emphasis. While emphasis is part of the content of the document, its presentation may be in an oblique font style, but one need not necessarily imply the other: for example, emphasis in text that is already oblique should in fact be printed in some other font style or weight (e.g., normal again or in boldface) that is distinct from that of the text it appears in; conversely, oblique text need not imply emphasis—it could be used to style a preface, for example. Moreover, notice that the foregoing sentence assumes that the output mode is visual; but if the mode were audio, the indication of the intended meaning (emphasis) could be achieved by formatting the `<em>`...`</em>` words with a louder or higher-pitched voice by the text-to-speech synthesizer. Separation of content and presentation means that markup elements are always used for document structure, never for presentation (which is properly the job of tools such as XSLT, XSL, and CSS).

Machine readability

The general aim of separation of presentation and content is **machine readability**, that is, making it possible for machines to detect meaning or intent. (The machine readability is then a means to desired ends, as discussed below.) For example, a human being reading a document has little trouble to grab from context that an oblique rendering in one place would be emphasized text, but in another place is a title of a book. However, as robots and crawlers have more difficulty with this task, separation of presentation and content generally aids in their distinguishing of such things which are presented in the same way, but have a different meaning—or have the same meaning, but are presented in a different way.

Machine readability allows affordably serving the information to a wider variety of users (in a presentation they can understand), where users may be humans or machines. This requires the ability to recast abstractions in new instances quickly and cheaply (that is, without time-consuming reworking), which generally requires automation rather than person-hours of labor. For example:

- The ability to deliver the same information in different media, and to change the medium quickly and cheaply; and within one medium, to change instances easily
- To serve the same message to different users:
  - as printed display (for technophobes or for users with contextual desire for print)
  - as printed display with the typesetting recast into various graphic designs without time being spent on any manual reworking of the content (a good example is given at [CSS Zen Garden](https://csszengarden.com))
  - as online visual display (for most users in most instances)
  - as online visual display in various graphic designs
  - as online audio (for blind users or for sighted users with contextual desire for audio)
  - as braille (for blind users)
  - as input to an API (for users that are machines)
The other machines can then take the information and do further transformations or actions. These may be ones that people cannot do (or cannot do quickly and cheaply), but machines can do (and can do more quickly and more cheaply). For example:

- Take a book and translate it into another natural language
- Take audio of speech and translate it into another natural language
- Take audio of speech and transcribe it for reading (for deaf users or for hearing users who want transcription of voicemail into e-mail or IM)
- Take the data contained in an entire library and search through it for ABC-XYZ, then turn every instance of ABC blue, and serve every instance of XYZ to a machine that will categorize it

5.4.2 Real-world applications

Excel tables

<table>
<thead>
<tr>
<th>Formatted as Table</th>
<th>Formatted as non-Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Formulas default to apply to all rows of a column.</td>
<td>• Formulas apply to selected cells only. Must manually apply to other rows of data range.</td>
</tr>
<tr>
<td>• Quick selection keys (i.e. Shift and Control) stop at table border, even if the last cell of the table row/column does not contain any value.</td>
<td>• Quick selection keys stops at next non-blank cell, or max row/column, whichever comes first.</td>
</tr>
<tr>
<td>• Option to format row/column header and banded rows/columns.</td>
<td></td>
</tr>
<tr>
<td>• Filters automatically applied to header rows.</td>
<td></td>
</tr>
<tr>
<td>• Direct referencing of the whole table from formulas.</td>
<td></td>
</tr>
<tr>
<td>• Adding a new row below or column to the right the last one automatically extends the table.</td>
<td></td>
</tr>
</tbody>
</table>

Document presentation

Use simple markup language (such as reStructuredText used in this very document) with simple, unobstructive syntax to easily create (often using generic text editor), and to read in its raw form. The same raw texts can be used to produce HTML pages like the one you are viewing, or PDF books with additional formatting/branding options for printing.

Going to the other side of the spectrum, typesetting, which is the composition of texts by means of arranging physical types or digital equivalents. TeX (and the more user-friendly LaTeX) are popular and powerful typesetting systems to systematically encode the structure of a document for presentation, accounting for ligature, pre-press, strut, typefaces, references, indexing etc. TeX/LaTeX are very widely used in academic circles for published papers and even books.
Word styles

If Microsoft Word must be used to draft a document, styles in Word save time and make your document look good. Common styles—title, subtitle, headings (multiple levels), etc.—can be outfitted with font name, size, color, paragraph alignment and spacing, and many more formatting characteristics. Once assigned to texts, style changes apply to all associated texts, instead of applying the same changes to multiple texts.

Styles can also contain body structure definition to allow quick navigation of the whole document.

AIS presentations
An MLM can be triggered when users open the Order Entry Worksheet from the toolbar icon or the GoTo menu, to change the default session type and other details of the order entry session. The MLM is not triggered when the Order Entry Worksheet is opened from within another process such as reconciling.

Should St. Vincent utilize the MLM to change the default session type and other details of the order entry session when the Order Entry Worksheet is opened?

- **Utilize this feature.**
  - Potential to avoid order entry in Standard session (as opposed to HOLD) for Pre-Admit visits.
  - Potential consolidation with existing customization "STAT if ER".
  - Are there other use cases?
  - It may be beneficial to ask for feedbacks from St. Vincent ministries before development.

- **Do not utilize this feature.**
Miscellaneous

Miscellaneous items not best suited for other categories. Section author: Le Yang <le.yang@leidoshealth.com>, Marcia Anderson <marcia.anderson@ascension.org>

6.1 Optimizations

- not possible; need help; need vendor engagement
- solution likely exists; need additional research, follow-up, or proof of concept; may use some customization
- solution exists; quick to implement
- remediated; solution implemented

Contents

- Optimizations
  - PHASE I
  - PHASE II

6.1.1 PHASE I

Last Does Taken Date and Time are required before Outpatient Medication Review (OMR) can save as complete.  

Change EP setting Rx Writer|Med Review|MedReviewLastDoseTakenRequired from TRUE to FALSE will make Last Does Taken Date and Time optional when saving the OMR as complete.
Fig. 6.1: Last Dose Taken is not required when entering new home medications
Fig. 6.2: User can save OMR as complete without filling in Last Dose Taken
Patient Status, Resuscitation Status, or Diet orders should not require the user to discontinue existing one first before entering new.

With the help of custom library made available via the Insulin Program, existing orders can be discontinued once replacement order is placed. Below are the recommended actions:

- Limited orders to consider by this process
  - *Patient Status group:* Patient Status | Patient Status - Behavioral Health | Patient Status - Hematology/Oncology
  - Resuscitation Status
  - *Diets group:* Diet | Diet, Bariatric | NPO

- Use Duplicate Checking Policy to define message type and action.
  - Add to system data item `HVC-DuplicateMsgType` new message type to mean “replace current order by d/c existing”;
  - Add to system data item `HVC-Duplicate` new class type values to group orders together;
  - Add to standard (STD_DUPLICATE) and supporting MLMs to implement “replace current order by d/c existing”;

Weekly lab should be 0900 and then let user pick a day

**Note:** Is it possible to programmatically update order repeat property?

Lab tests are defaulting to “new” versus “tomorrow AM”

**Note:** Need more information on this one

Sort documents by type, not author; Sort results by report by order

Default viewing options for Documents and Results (i.e. Display Format) can be pushed out to users using the User Profile Tool. Need to determine:

- New Display Format with enterprise-defined sorting and display type can be pushed out to users using the User Profile Tool. Is it possible to set default Display Format?

- If not, what are the downsides to replace user profiles (by role) altogether? Can we use this opportunity to standardize other preferences such as Order Rec Toolbar, Toolbar icons, etc.?

Reaching out to the Security team to validate feasibility.
End users should only have access to the latest version of a document.

- Review ER quick launch documents
- Review document catalog

No collection source on lab results i.e. thoracentesis right or left.

Note: Need more information on this one

Culture lab orders have fields built on the form to indicate the source of culture collection, with different lists becoming available. The indicated collection source is available via the outbound order interface meaning it has potential to be interfaced back to SCM from Sunquest with results.

Fig. 6.3: Sample of collection source on Culture forms

Specimen source is included in the list of result components associated with Culture orders. While this may not correspond directly to the collection source, it shows possible places such data can be reviewed.

Fig. 6.4: Source shown on culture result

6.1. Optimizations
Apple request date and time to all orders on “common” order sets.

Add requestedDate and requestedTime to common order set forms. Map the fields to orders included in the order set.

Revisit document naming convention for easier identification of the documents, such as KBC or Hosp. Removing version numbers.

**Transcribed Document (interfaced)**

<table>
<thead>
<tr>
<th>Existing Name</th>
<th>Proposed Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hosp. Behavior Health Consultation Note</td>
<td>Behavior Health Consultation Note</td>
</tr>
<tr>
<td>Hosp. Behavior Health Discharge Summary</td>
<td>Behavior Health Discharge Summary</td>
</tr>
<tr>
<td>Hosp. Behavior Health History &amp; Physical</td>
<td>Behavior Health History &amp; Physical</td>
</tr>
<tr>
<td>Hosp. Card H &amp; P</td>
<td>Card H &amp; P</td>
</tr>
<tr>
<td>Hosp. Cardiac Cath</td>
<td>Cardiac Cath</td>
</tr>
<tr>
<td>Hosp. Cardiology Discharge Summary</td>
<td>Cardiology Discharge Summary</td>
</tr>
<tr>
<td>Hosp. Consult Oncology Report</td>
<td>Consult Oncology Report</td>
</tr>
<tr>
<td>Hosp. Consultation</td>
<td>Consultation</td>
</tr>
<tr>
<td>Hosp. Diagnostic Testing</td>
<td>Diagnostic Testing</td>
</tr>
<tr>
<td>Hosp. Discharge Summary</td>
<td>Discharge Summary</td>
</tr>
<tr>
<td>Hosp. Echocardiogram</td>
<td>Echocardiogram</td>
</tr>
<tr>
<td>Hosp. Electrophys. Study</td>
<td>Electrophys. Study</td>
</tr>
<tr>
<td>Hosp. Emergency Room Note</td>
<td>Emergency Room Note</td>
</tr>
<tr>
<td>Hosp. ENG/BSER/AER/Speech &amp; Aud</td>
<td>ENG/BSER/AER/Speech &amp; Aud</td>
</tr>
<tr>
<td>Hosp. Fishers Rad-Onc Note</td>
<td>Fishers Rad-Onc Note</td>
</tr>
<tr>
<td>Hosp. Head &amp; Neck/Tumor Center</td>
<td>Head &amp; Neck/Tumor Center</td>
</tr>
<tr>
<td>Hosp. History &amp; Physical</td>
<td>History &amp; Physical</td>
</tr>
<tr>
<td>Hosp. Neurological Results</td>
<td>Neurological Results</td>
</tr>
<tr>
<td>Hosp. Northside Carmel Echo</td>
<td>Northside Carmel Echo</td>
</tr>
<tr>
<td>Hosp. Novalis Rad-Onc Note</td>
<td>Novalis Rad-Onc Note</td>
</tr>
<tr>
<td>Hosp. Nuclear Cardiology</td>
<td>Nuclear Cardiology</td>
</tr>
<tr>
<td>Hosp. Outpatient Rad/Onc Center</td>
<td>Outpatient Rad/Onc Center</td>
</tr>
<tr>
<td>Hosp. Pediatric Discharge Summary</td>
<td>Pediatric Discharge Summary</td>
</tr>
<tr>
<td>Hosp. Pediatric Echo</td>
<td>Pediatric Echo</td>
</tr>
<tr>
<td>Hosp. Pediatric History &amp; Physical</td>
<td>Pediatric History &amp; Physical</td>
</tr>
<tr>
<td>Hosp. Peds ER</td>
<td>Peds ER</td>
</tr>
<tr>
<td>Hosp. Physical / Occupational Therapy</td>
<td>Physical / Occupational Therapy</td>
</tr>
<tr>
<td>Hosp. Procedure Note</td>
<td>Procedure Note</td>
</tr>
<tr>
<td>Hosp. Radiation Oncology</td>
<td>Radiation Oncology</td>
</tr>
<tr>
<td>Hosp. Radiation Oncology/Letter</td>
<td>Radiation Oncology/Letter</td>
</tr>
</tbody>
</table>

Continued on next page
Table 6.1 – continued from previous page

<table>
<thead>
<tr>
<th>Existing Name</th>
<th>Proposed Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hosp. Sleep Study</td>
<td>Sleep Study</td>
</tr>
<tr>
<td>Hosp. Speech Therapy</td>
<td>Speech Therapy</td>
</tr>
<tr>
<td>Hosp. Speech Therapy Naab Rd</td>
<td>Speech Therapy Naab Rd</td>
</tr>
<tr>
<td>Hosp. Transesophageal Echo</td>
<td>Transesophageal Echo</td>
</tr>
<tr>
<td>Hosp. Treadmill</td>
<td>Treadmill</td>
</tr>
<tr>
<td>Hosp. Women’s Consultation Note</td>
<td>Women’s Consultation Note</td>
</tr>
<tr>
<td>Hosp. Women’s Discharge Note</td>
<td>Women’s Discharge Note</td>
</tr>
<tr>
<td>Hosp. Women’s Procedure Note</td>
<td>Women’s Procedure Note</td>
</tr>
<tr>
<td>Hosp. Wound Center Note</td>
<td>Wound Center Note</td>
</tr>
<tr>
<td>Perioperative - IntraOp Note</td>
<td>Perioperative - IntraOp Note</td>
</tr>
<tr>
<td>Perioperative - Phase II Recovery Note</td>
<td>Perioperative - Phase II Recovery Note</td>
</tr>
<tr>
<td>Perioperative - PostOp Note</td>
<td>Perioperative - PostOp Note</td>
</tr>
<tr>
<td>Perioperative - Pre Admit Note</td>
<td>Perioperative - Pre Admit Note</td>
</tr>
<tr>
<td>Perioperative - PreOp Note</td>
<td>Perioperative - PreOp Note</td>
</tr>
</tbody>
</table>

Structured Notes and Flowsheets (non-interfaced)

<table>
<thead>
<tr>
<th>Document Type</th>
<th>Existing Name</th>
<th>Proposed Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flowsheet (Adult)</td>
<td>1. KBC Adult Plan of Care 3.4</td>
<td>1. Adult Plan of Care</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>2. Intake Output - Adult 3.4</td>
<td>2. Intake Output - Adult</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>2. KBC Adult Assessment Intervention 3.4</td>
<td>2. Adult Assessment Intervention</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>2. KBC Behavioral Health Flowsheet 3.4</td>
<td>2. Behavioral Health Flowsheet</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>3. Patient Care Flowsheet 3.4</td>
<td>3. Patient Care Flowsheet</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>Blood Transfusion FS v.3</td>
<td>Blood Transfusion FS</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>CV Procedure 3.4</td>
<td>CV Procedure</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>Chemotherapy_v2</td>
<td>Chemotherapy</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>Diabetes Flowsheet_v3</td>
<td>Diabetes Flowsheet</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>ECMO Flowsheet_v2</td>
<td>ECMO Flowsheet</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>ER Nursing Flowsheet 3.4</td>
<td>ER Nursing Flowsheet</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>KBC Adult Hemodialysis Record 3.4</td>
<td>Adult Hemodialysis Record</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>KBC Adult Rehab Record 3.4</td>
<td>Adult Rehab Record</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>KBC CPAP BiPAP Record 3.4</td>
<td>CPAP BiPAP Record</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>KBC Nutrition Assessment 3.4</td>
<td>Nutrition Assessment</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>KBC Ventilator Record Version 3.4 V2</td>
<td>Ventilator Record Version</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>Observation Flowsheet 3.4</td>
<td>Observation Flowsheet</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>Post Discharge Follow Up FS</td>
<td>Post Discharge Follow Up FS</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>Post Procedure 3.4</td>
<td>Post Procedure</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>Procedure Sedation 3.4</td>
<td>Procedure Sedation</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>Spinal Cord Flowsheet_v3</td>
<td>Spinal Cord Flowsheet</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>Swing Bed Assessment 3.4</td>
<td>Swing Bed Assessment</td>
</tr>
<tr>
<td>Flowsheet (Adult)</td>
<td>Therapy Progress Note (Behavioral Health)</td>
<td>Therapy Progress Note (Behavioral Health)</td>
</tr>
</tbody>
</table>

6.1. Optimizations
### Table 6.2 – continued from previous page

<table>
<thead>
<tr>
<th>Document Type</th>
<th>Existing Name</th>
<th>Proposed Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flowsheet (Pediatric)</td>
<td>1. KBC Newborn Plan of Care 3.4</td>
<td>1. Newborn Plan of Care</td>
</tr>
<tr>
<td>Flowsheet (Pediatric)</td>
<td>1. KBC NICU Plan of Care 3.4</td>
<td>1. NICU Plan of Care</td>
</tr>
<tr>
<td>Flowsheet (Pediatric)</td>
<td>2. KBC Newborn Assessment Intervention 3.4</td>
<td>2. Newborn Assessment Intervention</td>
</tr>
<tr>
<td>Flowsheet (Pediatric)</td>
<td>2. KBC NICU Assessment Intervention 3.4</td>
<td>2. NICU Assessment Intervention</td>
</tr>
<tr>
<td>Flowsheet (Pediatric)</td>
<td>2. NICU/NBN I&amp;O Flowsheet</td>
<td>2. NICU/NBN I&amp;O Flowsheet</td>
</tr>
<tr>
<td>Structured Note</td>
<td>NICU-NBN Procedure 3.4</td>
<td>NICU-NBN Procedure</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Acute Physical Rehab Center Plan for Continued Care</td>
<td>Acute Physical Rehab Center Plan for Continued Care</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Acute Rehab Home Assessment</td>
<td>Acute Rehab Home Assessment</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ARU Individualized Overall Plan Of Care</td>
<td>ARU Individualized Overall Plan Of Care</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ARU Weekly Team Conference</td>
<td>ARU Weekly Team Conference</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Asthma Discharge Instructions (Peds)</td>
<td>Asthma Discharge Instructions</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Baclofen Trial Assessment</td>
<td>Baclofen Trial Assessment</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Berlin Questionnaire Sleep Evaluation</td>
<td>Berlin Questionnaire Sleep Evaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Care Conference Note</td>
<td>Care Conference Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Case Management SN</td>
<td>Case Management SN</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Child Abuse/Neglect Evaluation Note</td>
<td>Child Abuse/Neglect Evaluation Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Clinical Nurse Specialist</td>
<td>Clinical Nurse Specialist</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Clinical Psychologist Note</td>
<td>Clinical Psychologist Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>CV Pre Procedure Short Note</td>
<td>CV Pre Procedure Short Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>CV Pre Procedure Note 3.4</td>
<td>CV Pre Procedure Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Death Note v2</td>
<td>Death Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Diabetes Educator Note 3.4</td>
<td>Diabetes Educator Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Discharge From Behavioral Health</td>
<td>Discharge From Behavioral Health</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Discharge Note</td>
<td>Discharge Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Discharge Note - Pediatrics</td>
<td>Discharge Note - Pediatrics</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Downtime Note</td>
<td>Downtime Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Dysphagia Screen</td>
<td>Dysphagia Screen</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER Child Life/Peds ED Chart Note</td>
<td>ER Child Life/Peds ED Chart Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER Code Note</td>
<td>ER Code Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER MD Admit Disposition</td>
<td>ER MD Admit Disposition</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER MD Discharge Disposition</td>
<td>ER MD Discharge Disposition</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER MD Discharge Disposition v2</td>
<td>ER MD Discharge Disposition v2</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER Nursing Discharge</td>
<td>ER Nursing Discharge</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER Nursing Documentation v3</td>
<td>ER Nursing Documentation v3</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER Nursing Flowsheet 3.4</td>
<td>ER Nursing Flowsheet 3.4</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER Nursing Intervention/Reassessment</td>
<td>ER Nursing Intervention/Reassessment</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER Pediatric Nursing Documentation v3</td>
<td>ER Pediatric Nursing Documentation v3</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER PEDS Physician Documentation</td>
<td>ER PEDS Physician Documentation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER Physician Addendum Note</td>
<td>ER Physician Addendum Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER Physician Exam - Pediatrics</td>
<td>ER Physician Exam - Pediatrics</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER Physician Exam v7</td>
<td>ER Physician Exam v7</td>
</tr>
<tr>
<td>Structured Note</td>
<td>ER Physician Exam v8 (ANDERSON ED ONLY - with Dragon)</td>
<td>ER Physician Exam v8 (ANDERSON ED ONLY - with Dragon)</td>
</tr>
</tbody>
</table>
Table 6.2 – continued from previous page

<table>
<thead>
<tr>
<th>Document Type</th>
<th>Existing Name</th>
<th>Proposed Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured Note</td>
<td>ER Triage Note v2</td>
<td>ER Triage Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Ethics Consult</td>
<td>Ethics Consult</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Ethics Follow Up</td>
<td>Ethics Follow Up</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Fall Structure Note</td>
<td>Fall Structure Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>General Note 3.4</td>
<td>General Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Heart Attack Discharge Instructions V2</td>
<td>Heart Attack Discharge Instructions</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Heart Failure Discharge Instructions V2</td>
<td>Heart Failure Discharge Instructions</td>
</tr>
<tr>
<td>Structured Note</td>
<td>History and Physical - Pediatrics v2</td>
<td>History and Physical - Pediatrics</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Influenza Vaccine Assessment</td>
<td>Influenza Vaccine Assessment</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Interdisciplinary Behavioral Health Assessment-Psychosocial</td>
<td>Interdisciplinary Behavioral Health Assessment-Psychosocial</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Interdisciplinary Note</td>
<td>Interdisciplinary Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC Adult Patient Profile 3.4</td>
<td>Adult Patient Profile</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC Birth Record 3.4</td>
<td>Birth Record</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC Newborn NICU Patient Profile 3.4</td>
<td>Newborn NICU Patient Profile</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC OT Discharge Summary 3.4</td>
<td>OT Discharge Summary</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC OT Initial Evaluation 3.4</td>
<td>OT Initial Evaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC OT Reevaluation 3.4</td>
<td>OT Reevaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC Pediatric Patient Profile 3.4</td>
<td>Pediatric Patient Profile</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC PT Discharge Summary 3.4</td>
<td>PT Discharge Summary</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC PT Initial Evaluation 3.4</td>
<td>PT Initial Evaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC PT Reevaluation 3.4</td>
<td>PT Reevaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC SLP Discharge Summary 3.4</td>
<td>SLP Discharge Summary</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC SLP Initial Evaluation 3.4</td>
<td>SLP Initial Evaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC SLP Reevaluation 3.4</td>
<td>SLP Reevaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>KBC Social Work Assessment 3.4</td>
<td>Social Work Assessment</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Lactation Note</td>
<td>Lactation Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Master Treatment Plan (Behavioral Health)</td>
<td>Master Treatment Plan (Behavioral Health)</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Minor Laser Procedure</td>
<td>Minor Laser Procedure</td>
</tr>
<tr>
<td>Structured Note</td>
<td>MRI Screening Form V2</td>
<td>MRI Screening Form</td>
</tr>
<tr>
<td>Structured Note</td>
<td>NICU Discharge Instructions</td>
<td>NICU Discharge Instructions</td>
</tr>
<tr>
<td>Structured Note</td>
<td>NICU-NBN Procedure 3.4</td>
<td>NICU-NBN Procedure</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Organ Donation Information for IOPO</td>
<td>Organ Donation Information for IOPO</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Ostomy Evaluation (WOC)</td>
<td>Ostomy Evaluation (WOC)</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Pediatric ARU Weekly Team Conference</td>
<td>Pediatric ARU Weekly Team Conference</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Pediatric Influenza Vaccine Assessment</td>
<td>Pediatric Influenza Vaccine Assessment</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Pediatric Pneumococcal Vaccine Assessment</td>
<td>Pediatric Pneumococcal Vaccine Assessment</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Pediatric Progress Note</td>
<td>Pediatric Progress Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Pharmacy Note_v2</td>
<td>Pharmacy Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>PICC Insertion Note</td>
<td>PICC Insertion Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Pneumococcal Vaccine Assessment</td>
<td>Pneumococcal Vaccine Assessment</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Post Discharge Follow Up</td>
<td>Post Discharge Follow Up</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Pre-op/Peri-op Document 3.4</td>
<td>Pre-op/Peri-op Document</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Pre-Procedure Note 3.4</td>
<td>Pre-Procedure Note</td>
</tr>
</tbody>
</table>
### Table 6.2 – continued from previous page

<table>
<thead>
<tr>
<th>Document Type</th>
<th>Existing Name</th>
<th>Proposed Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured Note</td>
<td>Progress Note</td>
<td>Progress Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Progress Note - Communication- all clinicians</td>
<td>Progress Note - Communication- all clinicians</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Progress Note - Daily (Daily Progress Note)</td>
<td>Progress Note - Daily (Daily Progress Note)</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Progress Note - Final (Final Progress Note)</td>
<td>Progress Note - Final (Final Progress Note)</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Progress Note - Procedural or Post-Operative</td>
<td>Progress Note - Procedural or Post-Operative</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Psychosocial Assessment (Behavioral Health)</td>
<td>Psychosocial Assessment (Behavioral Health)</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Rehab Restorative Aid Note</td>
<td>Rehab Restorative Aid Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Release of Information</td>
<td>Release of Information</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Respiratory Care Bronchodilator Evaluation</td>
<td>Respiratory Care Bronchodilator Evaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Respiratory Care Pre-Surgical Consultation</td>
<td>Respiratory Care Pre-Surgical Consultation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Social Services Note</td>
<td>Social Services Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Social Work Initial Assessment</td>
<td>Social Work Initial Assessment</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Speech Therapy Pediatric Initial Evaluation</td>
<td>Speech Therapy Pediatric Initial Evaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Speech Therapy Peds Discharge Evaluation</td>
<td>Speech Therapy Peds Discharge Evaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Stroke Discharge Instructions</td>
<td>Stroke Discharge Instructions</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Surgery / Procedure Note</td>
<td>Surgery / Procedure Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Surgery Record - Ophthalmology</td>
<td>Surgery Record - Ophthalmology</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Surgery Record v3</td>
<td>Surgery Record v3</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Swallowing - Bedside Discharge Evaluation</td>
<td>Swallowing - Bedside Discharge Evaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Swallowing - Bedside Evaluation</td>
<td>Swallowing - Bedside Evaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Swallowing - FEES Evaluation</td>
<td>Swallowing - FEES Evaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Swallowing - VFSS Discharge Evaluation</td>
<td>Swallowing - VFSS Discharge Evaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Swallowing - VFSS Evaluation</td>
<td>Swallowing - VFSS Evaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Swallowing - VFSS Preliminary Note</td>
<td>Swallowing - VFSS Preliminary Note</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Swallowing - VFSS Preliminary Report</td>
<td>Swallowing - VFSS Preliminary Report</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Swing Bed PHP 3.4</td>
<td>Swing Bed PHP 3.4</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Therapeutic Recreation Mental Health Evaluation</td>
<td>Therapeutic Recreation Mental Health Evaluation</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Transfer Note v2</td>
<td>Transfer Note v2</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Weekly Wound Measurement Note (Rehab)</td>
<td>Weekly Wound Measurement Note (Rehab)</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Wound Discharge Summary</td>
<td>Wound Discharge Summary</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Wound Evaluation (Rehab)</td>
<td>Wound Evaluation (Rehab)</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Wound Progress Note (Rehab)</td>
<td>Wound Progress Note (Rehab)</td>
</tr>
<tr>
<td>Structured Note</td>
<td>Wound/Skin Evaluation (WOC Team)</td>
<td>Wound/Skin Evaluation (WOC Team)</td>
</tr>
</tbody>
</table>

### 6.1.2 PHASE II

#### Discharge process

Use Workflow Manager to facilitate and guide clinicals to complete required and optional steps of the discharge process.
6.2 Identification of Research Patients

6.2.1 Requirements

Phase 1:

- Need the ability to flag research patients from CTMS to Athena and SCM
  - Flag the patient across all visits in both systems and the ability to turn flag off when patient off of the study
  - Most likely an interface message from CTMS to Athena and SCM
  - Triggers need defined
  - Patients may be participating in more than one study
- Need common ID number to reference for entering information into CTMS product:
  - Current patient in Athena only
  - Current patient in SCM only
  - Research patient in neither system
  - There is currently a common ID between SCM and Athena – EE-EAD/Corp ID
- Research RNs need access to one or both systems as well as CTMS product
- Education plan for deployment of project (CNO/CLIC/CI’s)

Phase 2:

- Need ability to view and see consents and information on the project, consents, meds, via link to CTMS or interfaced document with the basics in Athena and or SCM

Important: Input needed from the Research team

- Available research names and codes (if relevant)
- Are Registration systems involved with the update communication?
- How will CTMS obtain EE-EAD for patients under research?

6.2.2 Implementation (Business)

Warning: General assumptions:
- Patient can participate in more than one research at a time i.e. allow multiple.
- CTMS has the ability to send interface updates both when patient is put on a study, and when taken off a study.
Update research patients in SCM manually or from CTMS

With sufficient privilege research patients can always be updated manually in SCM with all

If CTMS is to interface with SCM directly

If Registration systems are not involved with the communication between CTMS and SCM,

- Review current Advanced Visit List definition on Health Issue. Modify to accommodate the need of the research project.

Todo

Update MLM/AVL definition; come up with a list of data elements to capture for the AVL

Todo

Add “RES” (or “R”, or “RCH”, or “RESCH”) for “Research” on the header

Questions

What happens when patient is only available in CTMS or Athena, without any visit instance in SCM?

Create patient only using EE-EAD (A28). From Allscripts manual:: A28—Add person information (B/G)
For Registration, if the PV1 segment is sent, the message is processed the same as an A01. The applicable scenarios are as follows. > If no PV1 segment is sent, any patient information configured as visit scope (address information, contacts) is created with a general scope. > If no PV1 segment is sent, the application does not process enterprise-defined Column Data. > For Enterprise Registration, PV1 and PV2 segments are ignored and all data is stored at the patient level.

Note:  Let to test and validate proof of concept

Stakeholders

Tina Noonan - Research director at St. V technical - TBD Arvind Manoharan - Huron Manager (CTMS vendor) Simon Demertzis - Huron

Questions

1. Is there a need to identify research patients post patient discharge? (specific to AVL timing requirement)
2. Is there plan to identify such information during registration?
6.2.3 Task lists

- Technical
  - Interface (A28) - Le
  - AVL - Dyan and Le
  - Header - Dyan

- Implementation
  - Health Issue dict - Dyan
  - AVL & MLM - Le
  - Interface - TBD
  - Interface specification

- Documentation - Dyna and Le
  - Workflow diagrams - Le

Using patient list column flag
7.1 About these documents

These documents are generated from reStructuredText sources by Sphinx, a document processor specifically written for the Python documentation.

Contribution to these documents are essential to the continuous success and strength of the project.

7.1.1 PDP Purpose and Guideline

What is a PDP?

PDP stands for Program Documentation Proposal. A PDP is a design document providing information to the team, or describing a new feature or its processes or environment. The PDP should provide a concise business or technical specification of the feature.

We intend PDPs to be the primary mechanisms for document major new features, for collecting team input on an issue, and for documenting the design decisions that have gone into the program. The PDP author is responsible for building consensus within the community and documenting dissenting opinions.

Because the PDPs are maintained as text files in a versioned repository, their revision history is the historical record of the documentation.

Types of Documentation

- A Knowledge Base (KB) article outlines known issues and remediation steps.
- A Process document describes a process or workflow at an abstract level. While it is reasonable to include specific behaviors of system components, it should not include build and configuration details (these belong to the Configuration section).
- A Configuration document outlines detailed configuration steps for...
- An Informational document
PDP Metadata

Service
All, Cardiovascular, Trauma, ... (refer to STV listing)

Function Medication Management, Patient Care, Core, External, Orders

Type Process, Configuration, Knowledge Base, Informational

Module
- Orders
- MLM
- Customization
- Clinical Documentation

PDP workflow

PDP editors

The PDP editors are individuals responsible for managing the administrative and editorial aspects of the PDPs (e.g. adding to the repository and formatting). The current editors are:
- Le Yang
- ab
- de

Submitting a PDP

There is no information too small or trivial to include in the documentation. It is highly recommended that a single submission is focused to a specific topic, adhering to the current documentation structure. Small enhancements or changes often don’t need to initiate the document review process and can be requested by adding a ticket.

Each document submission must have a champion – someone who writes the documentation using the style and format described below, shepherds the discussions in the appropriate forums, and attempts to obtain the relevant information forming the documentation.

Peer review

Once the editor determines a PDP should add to the documentation it pass to peer review...
PDP maintenance

Once a PDP passes peer review, editors with push privilege will incorporate the PDP to the documentation repository. (more to come)

PDP formats and templates

There are two PDP formats available to authors: plaintext and reStructuredText. Both are UTF-8-encoded text files.

Plaintext PEPs are written with minimal structural markup that adheres to a rigid style. Sample Plaintext PDP Template contains instructions and a template you can use to get started writing your plaintext PDP.

ReStructuredText PDPs allow for rich markup that is still quite easy to read, but results in much better-looking and more functional HTML. Sample reStructuredText PDP Template contains instructions and a template for reStructuredText PDPs.

Section author: Le Yang <le.yang@leidoshealth.com>

7.1.2 Sample reStructuredText PDP Template

<table>
<thead>
<tr>
<th>Service</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Patient Care</td>
</tr>
<tr>
<td>Type</td>
<td>Process</td>
</tr>
<tr>
<td>Module</td>
<td>N/A</td>
</tr>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
</tbody>
</table>

Abstract

This PDP provides a boilerplate or sample template for creating your own reStructuredText PEPs. In conjunction with the content guidelines in PDP Purpose and Guideline, this should make it easy for you to conform your own PEPs to the format outlined below.

Note: if you are reading this template via the web, you should first grab the text (reStructuredText) source of this PDP in order to complete the steps below. DO NOT USE THE HTML FILE AS YOUR TEMPLATE!

The source for this (or any) template can be downloaded here.

If you would prefer not to use markup in your PDP, please see Sample Plaintext PDP Template.

Rationale

PDP submissions come in a wide variety of forms, not all adhering to the format guidelines set forth below. Use this template, in conjunction with the format guidelines below, to ensure that your PDP submission won’t get automatically rejected because of form.

7.1. About these documents

89
ReStructuredText is offered as an alternative to plaintext PEPs, to allow PDP authors more functionality and expressivity, while maintaining easy readability in the source text. The processed HTML form makes the functionality accessible to readers: live hyperlinks, styled text, tables, images, and automatic tables of contents, among other advantages.

How to Use This Template

To use this template you must first decide the PDP type. When in doubt, read PDP Purpose and Guideline for details.

Once you’ve decided which type of PDP yours is going to be, follow the directions below.

• Make a copy of this file (.txt file, not HTML!) or downloaded here and perform the following edits.

• Replace the sectionauthor directive with your name, and optionally your email address. Be sure to follow the format carefully: your name must appear first, and it must not be contained in parentheses. Your email address may appear second (or it can be omitted) and if it appears, it must appear in angle brackets. It is okay to obfuscate your email address.

• Change the Service, Function, and Type header to the service applicable to the PDP.

• Change the Title to the title of your PDP.

• Change the Status header to “Draft”.

• Change the Service header to the applicable service lie, or “All” if it does not apply to specific service line.

• Change the Function and Type header to one of the functions described in PDP Metadata.

• Now write your Abstract, Rationale, and other content for your PDP, replacing all this gobbledygook with your own text. Be sure to adhere to the format guidelines below, specifically on the prohibition of tab characters and the indentation requirements.

• Send your PDP submission to the PDP editors.

ReStructuredText PDP Formatting Requirements

The following is a PDP-specific summary of reStructuredText syntax. For the sake of simplicity and brevity, much detail is omitted. For more detail, see Resources below. Literal blocks (in which no markup processing is done) are used for examples throughout, to illustrate the plaintext markup.

General

Tab characters must never appear in the document at all.
Section Headings

PDP headings must begin in column zero and the initial letter of each word must be capitalized as in book titles. Acronyms should be in all capitals. Section titles must be adorned with an underline, a single repeated punctuation character, which begins in column zero and must extend at least as far as the right edge of the title text (4 characters minimum). Title texts are verline/underline-adorned with “#” (pound signs). First-level section titles are underlined with “=” (equals signs), second-level section titles with “-” (hyphens), third-level section titles with “*” (asterisk), fourth-level section titles with “^” (caret), and fifth-level section titles with “’’” (single quotes or apostrophes). For example:

```
****************************
Title of PDP (Title Case)
****************************
First-level section (sentence case)
===============================
Second-level section
---------------------
Third-level section
***********
Fourth-level section
^^^^^^^^^^
Fifth-level section
'''''''''
```

You shouldn’t have more than five levels of sections in your PDP. If you do, you should consider rewriting it.

You must use two blank lines between the last line of a section’s body and the next section heading. If a subsection heading immediately follows a section heading, a single blank line in-between is sufficient.

The body of each section is not normally indented, although some constructs do use indentation, as described below. Blank lines are used to separate constructs.

Paragraphs

Paragraphs are left-aligned text blocks separated by blank lines. Paragraphs are not indented unless they are part of an indented construct (such as a block quote or a list item).

Inline Markup

Portions of text within paragraphs and other text blocks may be styled. For example:

```
Text may be marked as *emphasized* (single asterisk markup, typically shown in italics) or **strongly emphasized** (double asterisks, typically boldface). ```Inline literals``` (using double
```
backquotes) are typically rendered in a monospaced typeface. No further markup recognition is done within the double backquotes, so they're safe for any kind of code snippets.

Block Quotes

Block quotes consist of indented body elements. For example:

This is a paragraph.

   This is a block quote.

   A block quote may contain many paragraphs.

Block quotes are used to quote extended passages from other sources. Block quotes may be nested inside other body elements. Use 4 spaces per indent level.

Literal Blocks

Literal blocks are used for code samples or preformatted ASCII art. To indicate a literal block, preface the indented text block with “::” (two colons). The literal block continues until the end of the indentation. Indent the text block by 4 spaces. For example:

This is a typical paragraph. A literal block follows.

::

   for a in [5,4,3,2,1]: # this is program code, shown as-is
       print a
   print "it's..."
   # a literal block continues until the indentation ends

The paragraph containing only “::” will be completely removed from the output; no empty paragraph will remain. “::” is also recognized at the end of any paragraph. If immediately preceded by whitespace, both colons will be removed from the output. When text immediately precedes the “::”, one colon will be removed from the output, leaving only one colon visible (i.e., “::” will be replaced by “:”). For example, one colon will remain visible here:

Paragraph::

   Literal block

Lists

Bullet list items begin with one of “-”, “*”, or “+” (hyphen, asterisk, or plus sign), followed by whitespace and the list item body. List item bodies must be left-aligned and indented relative to the bullet; the text immediately after the bullet determines the indentation. For example:
This paragraph is followed by a list.

* This is the first bullet list item. The blank line above the first list item is required; blank lines between list items (such as below this paragraph) are optional.

* This is the first paragraph in the second item in the list.

This is the second paragraph in the second item in the list. The blank line above this paragraph is required. The left edge of this paragraph lines up with the paragraph above, both indented relative to the bullet.

- This is a sublist. The bullet lines up with the left edge of the text blocks above. A sublist is a new list so requires a blank line above and below.

* This is the third item of the main list.

This paragraph is not part of the list.

Enumerated (numbered) list items are similar, but use an enumerator instead of a bullet. Enumerators are numbers (1, 2, 3, ...), letters (A, B, C, ...; uppercase or lowercase), or Roman numerals (i, ii, iii, iv, ...; uppercase or lowercase), formatted with a period suffix (“1.”, “2.”), parentheses (“(1)”, “(2)”), or a right-parenthesis suffix (“1)”, “2)”). For example:

1. As with bullet list items, the left edge of paragraphs must align.

2. Each list item may contain multiple paragraphs, sublists, etc.

   This is the second paragraph of the second list item.

   a) Enumerated lists may be nested.

   b) Blank lines may be omitted between list items.

Definition lists are written like this:

what

Definition lists associate a term with a definition.

how

The term is a one-line phrase, and the definition is one or more paragraphs or body elements, indented relative to the term.

Tables

Simple tables are easy and compact:

<table>
<thead>
<tr>
<th>=====</th>
<th>=====</th>
<th>========</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A and B</td>
</tr>
</tbody>
</table>
There must be at least two columns in a table (to differentiate from section titles). Column spans use underlines of hyphens (“Inputs” spans the first two columns):

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>

Text in a first-column cell starts a new row. No text in the first column indicates a continuation line; the rest of the cells may consist of multiple lines. For example:

<table>
<thead>
<tr>
<th>col 1</th>
<th>col 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Second column of row 1.</td>
</tr>
<tr>
<td>2</td>
<td>Second column of row 2.</td>
</tr>
<tr>
<td>3</td>
<td>Second line of paragraph.</td>
</tr>
</tbody>
</table>

- Second item in bullet list (row 3, column 2).

Hyperlinks

When referencing an external web page in the body of a PDP, you should include the title of the page in the text, with either an inline hyperlink reference to the URL or a footnote reference (see Footnotes below). Do not include the URL in the body text of the PDP.

Hyperlink references use backquotes and a trailing underscore to mark up the reference text; backquotes are optional if the reference text is a single word. For example:

In this paragraph, we refer to the `Python web site`_.

An explicit target provides the URL. Put targets in a References section at the end of the PDP, or immediately after the reference. Hyperlink targets begin with two periods and a space (the “explicit markup start”), followed by a leading underscore, the reference text, a colon, and the URL (absolute or relative):
The reference text and the target text must match (although the match is case-insensitive and ignores differences in whitespace). Note that the underscore trails the reference text but precedes the target text. If you think of the underscore as a right-pointing arrow, it points away from the reference and toward the target.

The same mechanism can be used for internal references. Every unique section title implicitly defines an internal hyperlink target. We can make a link to the Abstract section like this:

Here is a hyperlink reference to the `Abstract` section. The backquotes are optional since the reference text is a single word; we can also just write: Abstract_.

Footnotes containing the URLs from external targets will be generated automatically at the end of the References section of the PDP, along with footnote references linking the reference text to the footnotes.

Text of the form “PDP x” or “RFC x” (where “x” is a number) will be linked automatically to the appropriate URLs.

Footnotes

Footnote references consist of a left square bracket, a number, a right square bracket, and a trailing underscore:

This sentence ends with a footnote reference [1]_.

Whitespace must precede the footnote reference. Leave a space between the footnote reference and the preceding word.

When referring to another PDP, include the PDP number in the body text, such as “PDP 1”. The title may optionally appear. Add a footnote reference following the title. For example:

Refer to PDP 1 [2]_ for more information.

Add a footnote that includes the PDP’s title and author. It may optionally include the explicit URL on a separate line, but only in the References section. Footnotes begin with “.. ” (the explicit markup start), followed by the footnote marker (no underscores), followed by the footnote body. For example:

References
==========

.. [2] PDP 1, "PDP Purpose and Guidelines", Warsaw, Hylton
(http://www.python.org/dev/peps/PDP-0001)

If you decide to provide an explicit URL for a PDP, please use this as the URL template:

http://www.python.org/dev/peps/PDP-xxxx

PDP numbers in URLs must be padded with zeros from the left, so as to be exactly 4 characters wide, however PDP numbers in the text are never padded.

During the course of developing your PDP, you may have to add, remove, and rearrange footnote references, possibly resulting in mismatched references, obsolete footnotes, and confusion. Auto-numbered footnotes
allow more freedom. Instead of a number, use a label of the form “#word”, where “word” is a mnemonic consisting of alphanumerics plus internal hyphens, underscores, and periods (no whitespace or other characters are allowed). For example:

Refer to PDP 1 [#PDP-1]_ for more information.

References
========

.. [#PDP-1] PDP 1, "PDP Purpose and Guidelines", Warsaw, Hylton

http://www.python.org/dev/peps/PDP-0001

Footnotes and footnote references will be numbered automatically, and the numbers will always match. Once a PDP is finalized, auto-numbered labels should be replaced by numbers for simplicity.

**Images**

If your PDP contains a diagram, you may include it in the processed output using the “image” directive:

.. image:: dataflow.png

Any browser-friendly graphics format is possible: .png, .jpeg, .gif, .tiff, etc.

Since this image will not be visible to readers of the PDP in source text form, you should consider including a description or ASCII art alternative, using a comment (below).

**Comments**

A comment block is an indented block of arbitrary text immediately following an explicit markup start: two periods and whitespace. Leave the ”..” on a line by itself to ensure that the comment is not misinterpreted as another explicit markup construct. Comments are not visible in the processed document. For the benefit of those reading your PDP in source form, please consider including a descriptions of or ASCII art alternatives to any images you include. For example:

.. image:: dataflow.png

.. Data flows from the input module, through the "black box" module, and finally into (and through) the output module.

The Emacs stanza at the bottom of this document is inside a comment.

**Escaping Mechanism**

reStructuredText uses backslashes ("\") to override the special meaning given to markup characters and get the literal characters themselves. To get a literal backslash, use an escaped backslash (“\\”). There are two contexts in which backslashes have no special meaning: literal blocks and inline literals (see *Inline*
In these contexts, no markup recognition is done, and a single backslash represents a literal backslash, without having to double up.

If you find that you need to use a backslash in your text, consider using inline literals or a literal block instead.

**Habits to Avoid**

Many programmers who are familiar with TeX often write quotation marks like this:

```
'single-quoted' or `double-quoted`
```

Backquotes are significant in reStructuredText, so this practice should be avoided. For ordinary text, use ordinary 'single-quotes' or “double-quotes”. For inline literal text (see Inline Markup above), use double-backquotes:

```
``literal text: in here, anything goes```
```

**Resources**

Many other constructs and variations are possible. For more details about the reStructuredText markup, in increasing order of thoroughness, please see:

- A ReStructuredText Primer, a gentle introduction.
- Quick reStructuredText, a users’ quick reference.
- reStructuredText Markup Specification, the final authority.

The processing of reStructuredText PEPs is done using Docutils. If you have a question or require assistance with reStructuredText or Docutils, please post a message to the Docutils-users mailing list. The Docutils project web site has more information.

**References**

7.1.3 Sample Plaintext PDP Template

Download: pdp-template.txt

```
PEP: 9
Title: Sample Plaintext PEP Template
Version: $Revision$
Last-Modified: $Date$
Author: Barry Warsaw <barry@python.org>
Status: Active
Type: Process
Content-Type: text/plain
Created: 14-Aug-2001
Post-History:
```

7.1. About these documents
Abstract

This PEP provides a boilerplate or sample template for creating your own plaintext PEPs. In conjunction with the content guidelines in PEP 1 [1], this should make it easy for you to conform your own PEPs to the format outlined below.

Note: if you are reading this PEP via the web, you should first grab the plaintext source of this PEP in order to complete the steps below. DO NOT USE THE HTML FILE AS YOUR TEMPLATE!

To get the source this (or any) PEP, look at the top of the HTML page and click on the date & time on the "Last-Modified" line. It is a link to the source text in the Python repository.

If you would prefer to use lightweight markup in your PEP, please see PEP 12, "Sample reStructuredText PEP Template" [2].

Rationale

PEP submissions come in a wide variety of forms, not all adhering to the format guidelines set forth below. Use this template, in conjunction with the content guidelines in PEP 1, to ensure that your PEP submission won't get automatically rejected because of form.

How to Use This Template

To use this template you must first decide whether your PEP is going to be an Informational or Standards Track PEP. Most PEPs are Standards Track because they propose a new feature for the Python language or standard library. When in doubt, read PEP 1 for details or contact the PEP editors <peps@python.org>.

Once you've decided which type of PEP yours is going to be, follow the directions below.

- Make a copy of this file (.txt file, not HTML!) and perform the following edits.
- Replace the "PEP: 9" header with "PEP: XXX" since you don't yet have a PEP number assignment.
- Change the Title header to the title of your PEP.
- Leave the Version and Last-Modified headers alone; we'll take care of those when we check your PEP into Python's Subversion repository. These headers consist of keywords ("Revision" and "Date" enclosed in "$"-signs) which are automatically expanded by the repository. Please do not edit the expanded date or revision text.
- Change the Author header to include your name, and optionally your email address. Be sure to follow the format carefully: your name must appear first, and it must not be contained in parentheses. Your email address may appear second (or it can be omitted) and if it appears, it must appear in angle brackets. It is okay to obfuscate your email address.

- If there is a mailing list for discussion of your new feature, add a Discussions-To header right after the Author header. You should not add a Discussions-To header if the mailing list to be used is either python-list@python.org or python-dev@python.org, or if discussions should be sent to you directly. Most Informational PEPs don't have a Discussions-To header.

- Change the Status header to "Draft".

- For Standards Track PEPs, change the Type header to "Standards Track".

- For Informational PEPs, change the Type header to "Informational".

- For Standards Track PEPs, if your feature depends on the acceptance of some other currently in-development PEP, add a Requires header right after the Type header. The value should be the PEP number of the PEP yours depends on. Don't add this header if your dependent feature is described in a Final PEP.

- Change the Created header to today's date. Be sure to follow the format carefully: it must be in dd-mmm-yyyy format, where the mmm is the 3 English letter month abbreviation, e.g. one of Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec.

- For Standards Track PEPs, after the Created header, add a Python-Version header and set the value to the next planned version of Python, i.e. the one your new feature will hopefully make its first appearance in. Do not use an alpha or beta release designation here. Thus, if the last version of Python was 2.2 alpha 1 and you're hoping to get your new feature into Python 2.2, set the header to:

  Python-Version: 2.2

- Leave Post-History alone for now; you'll add dates to this header each time you post your PEP to python-list@python.org or python-dev@python.org. E.g. if you posted your PEP to the lists on August 14, 2001 and September 3, 2001, the Post-History header would look like:


You must manually add new dates and check them in. If you don't have check-in privileges, send your changes to the PEP editor.
- Add a Replaces header if your PEP obsoletes an earlier PEP. The value of this header is the number of the PEP that your new PEP is replacing. Only add this header if the older PEP is in "final" form, i.e. is either Accepted, Final, or Rejected. You aren't replacing an older open PEP if you're submitting a competing idea.

- Now write your Abstract, Rationale, and other content for your PEP, replacing all this gobbledygook with your own text. Be sure to adhere to the format guidelines below, specifically on the prohibition of tab characters and the indentation requirements.

- Update your References and Copyright section. Usually you'll place your PEP into the public domain, in which case just leave the "Copyright" section alone. Alternatively, you can use the Open Publication License[3], but public domain is still strongly preferred.

- Leave the little Emacs turd at the end of this file alone, including the formfeed character ("^L", or \f).

- Send your PEP submission to the PEP editors (peps@python.org), along with $100k in unmarked pennies. (Just kidding, I wanted to see if you were still awake. :)

Plaintext PEP Formatting Requirements

PEP headings must begin in column zero and the initial letter of each word must be capitalized as in book titles. Acronyms should be in all capitals. The body of each section must be indented 4 spaces. Code samples inside body sections should be indented a further 4 spaces, and other indentation can be used as required to make the text readable. You must use two blank lines between the last line of a section's body and the next section heading.

You must adhere to the Emacs convention of adding two spaces at the end of every sentence. You should fill your paragraphs to column 70, but under no circumstances should your lines extend past column 79. If your code samples spill over column 79, you should rewrite them.

Tab characters must never appear in the document at all. A PEP should include the standard Emacs stanza included by example at the bottom of this PEP.

When referencing an external web page in the body of a PEP, you should include the title of the page in the text, with a footnote reference to the URL. Do not include the URL in the body text of the PEP. E.g.

Refer to the Python Language web site [1] for more details. ...
When referring to another PEP, include the PEP number in the body text, such as "PEP 1". The title may optionally appear. Add a footnote reference, a number in square brackets. The footnote body should include the PEP's title and author. It may optionally include the explicit URL on a separate line, but only in the References section. Note that the pep2html.py script will calculate URLs automatically. For example:

... Refer to PEP 1 [7] for more information about PEP style ...

References

[7] PEP 1, PEP Purpose and Guidelines, Warsaw, Hylton
http://www.python.org/dev/peps/pep-0001/

If you decide to provide an explicit URL for a PEP, please use this as the URL template:

http://www.python.org/dev/peps/pep-xxxx/

PEP numbers in URLs must be padded with zeros from the left, so as to be exactly 4 characters wide, however PEP numbers in the text are never padded.

References

[1] PEP 1, PEP Purpose and Guidelines, Warsaw, Hylton
http://www.python.org/dev/peps/pep-0001/

[2] PEP 12, Sample reStructuredText PEP Template, Goodger, Warsaw
http://www.python.org/dev/peps/pep-0012/


Copyright

This document has been placed in the public domain.

7.1.4 Further reading

• reStructuredText Primer
• Sphinx Markup Constructs
7.2 Glossary

**Arden Syntax**  A language used for representing and sharing medical knowledge.

**CDM**  Charge Description Master, a comprehensive listing of items billable to a hospital patients.

**Comma-Separated Value, CSV**  A comma-separated values file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

**Medical Logic Module (MLM)**  An independent unit in a health knowledge base. Each MLM contains maintenance information, links to other sources of knowledge, and enough logic to make a single health decision.

**Resuscitation Status**  The name of order to indicate elected resuscitation option and intervention options in the case of Limited Resuscitation.

**SCM**  Sunrise Clinical Manager

**Service Provider, SP**  Service Provider, or cost centers defined in Soarian.

**SID**  Soarian Financials charge item code.

**Soarian**  Typically refers to Soarian Financials, the next enterprise registration system to replace INVISON.

**SPSID**  Unique Soarian Financials charge code for a given service provider. This code is interfaced from ancillary systems (including SCM) to Soarian Financials.
A
Arden Syntax, 102
AvailabilityOverride
  stv-func-order-create command line option, 18

C
CareProviderGUID
  stv-func-order-create command line option, 18
  stv-func-order-discontinue command line option, 19
CDM, 102
Certifying Physician, 5
ClientInfoObj
  form-func-substitute command line option, 17
ClientVisitGUID
  stv-func-order-create command line option, 18
CommunicationObj
  Comma-Separated Value, 102
  form-func-substitute command line option, 17
CSV, 102

D
DiscontinueDate
  stv-func-order-discontinue command line option, 19
DiscontinueTime
  stv-func-order-discontinue command line option, 19

E
ErrorMessage
  stv-func-order-create command line option, 18
  stv-func-order-discontinue command line option, 19

F
form-func-substitute command line option
  ClientInfoObj, 17
  CommunicationObj, 17
  FormObj, 17
  FormObj
    form-func-substitute command line option, 17

I
Inpatient Order, 6

M
Medical Logic Module (MLM), 102

O
OrderDefinitionObj
  stv-func-order-create command line option, 18
OrderGUID
  stv-func-order-discontinue command line option, 19

R
Reason
  stv-func-order-create command line option, 18
  stv-func-order-discontinue command line option, 19
ReferenceOrderGUID
  stv-func-order-create command line option, 18
Resuscitation Status, 102

S
SCM, 102
Service Provider, 102
SessionReason
  stv-func-order-create command line option, 18
  SessionType
stv-func-order-create command line option, 18
SID, 102
Soarian, 102
Source
  stv-func-order-create command line option, 18
  stv-func-order-discontinue command line option, 19
SP, 102
SPSID, 102
stv-func-order-create command line option
  AvailabilityOverride, 18
  CareProviderGUID, 18
  ClientVisitGUID, 18
  ErrorMessage, 18
  OrderDefinitionObj, 18
  Reason, 18
  ReferenceOrderGUID, 18
  SessionReason, 18
  SessionType, 18
  Source, 18
  SuppressNotification, 18
  UserDataObjList, 18
stv-func-order-discontinue command line option
  CareProviderGUID, 19
  DiscontinueDate, 19
  DiscontinueTime, 19
  ErrorMessage, 19
  OrderGUID, 19
  Reason, 19
  Source, 19
  SuppressNotification
    stv-func-order-create command line option, 18
U
UserDataObjList
  stv-func-order-create command line option, 18