

Membuat Produk Zope Sendiri

Belum ada produk Zope yang dapat memenuhi kebutuhan Anda? Tertantang untuk membuat sendiri?

Zope adalah *framework* pembuatan aplikasi web yang sangat besar dan luar biasa. Dengan Zope, banyak produk yang telah dibangun untuk mempermudah pembuatan aplikasi web yang luar biasa. Contoh produk Zope yang dapat digunakan untuk pembuatan aplikasi web adalah DTML dan ZPT. Dengan produk-produk tersebut, aplikasi-aplikasi web nan indah dan skalabel siap untuk lahir dari tangan Anda. Saat ini, apabila Anda melakukan instalasi Zope *default*, Anda telah dilengkapi dengan belasan produk siap pakai.

DTML, ZPT, dan Python Script mungkin adalah contoh-contoh produk luar biasa, yang dapat digunakan untuk membangun aplikasi web. Dengan menggunakan DTML atau ZPT saja, dan dilengkapi dengan Python Script untuk menangani pemrosesan data, Anda dapat membangun aplikasi dengan cepat dan mudah. Kemudian, apabila Anda memerlukan aplikasi *content management system* (CMS), produk CMF dapat langsung Anda gunakan. Ada lagi Plone, juga sebuah CMS, namun datang dengan tampilan yang indah luar biasa. Dan masih terdapat sangat banyak produk Zope lainnya. Di situs Zope, kita dapat menemukan ratusan produk Zope siap pakai, mulai yang sederhana sampai yang siap digunakan untuk lingkungan produktif.

Katakanlah, Anda adalah pemilik koran *online*. Kebutuhan utama Anda secara sederhana mungkin hanyalah menampilkan berita-berita yang telah dibagi dalam kategori-kategori tertentu. Dan secara sederhana pula, setiap wartawan Anda dapat langsung memasukkan berita tanpa melalui aneka sistem *workflow* yang rumit. Sederhana

memang. Untuk kebutuhan seperti itu, Anda dapat menggunakan CMF ataupun Plone. Tapi kedua produk Zope tersebut bukanlah produk yang sederhana. Mereka adalah produk yang kompleks. Dan Anda tidak butuh produk sekompleks itu. Anda butuh yang sederhana. Dengan Zope, Anda dapat membuatnya!

Produk Zope sendiri dapat dianalogikan seperti halnya sistem *plug-in* pada *web browser* kita. Web browser adalah aplikasi utama dengan fungsi yang telah jelas dan *plug-in* hanyalah sebagai pelengkap dengan fungsi yang juga spesifik. Bedanya dengan Zope, alih-alih menemukan sebuah aplikasi utama, kita hanya akan menemukan sebuah *framework*. *Framework* tersebut dilengkapi dengan berbagai '*plug-in*' yang lebih dikenal sebagai produk. Produk Zope sendiri, seperti telah disebutkan sebelumnya, dapatlah sangat sederhana. Dapat pula menjadi begitu kompleks.

Tentu saja tidak bijak apabila kita hantam kromo untuk membangun produk Zope tanpa melihat apakah produk yang tersedia saat ini telah mampu memenuhi kebutuhan kita, atau paling tidak, memenuhi sebagian kebutuhan kita. Dengan koleksi ratusan produk, mungkin salah satunya dapat dikembangkan lebih lanjut. Dan untuk itu, kita perlu mengetahui cara-cara pembuatan produk. Artikel ini akan membahas bagaimana kita membangun produk yang sederhana namun dapat digunakan. Tampilan produk yang kita buat pun akan tampak seprofesional produk-produk Zope lainnya. Mari sambut produk *Hello World*!

Sebelum kita memulai, ada baiknya bagi kita untuk memahami penggunaan



Python terlebih dahulu. Ya, pemahaman dasar-dasar Python adalah mutlak dalam pembuatan produk Zope. Kemudian, sebuah Zope yang telah berfungsi dengan baik pun harus tersedia di sistem kita. Dan jangan lupa, untuk mempermudah hidup dan mengurangi sakit mata, siapkan juga editor teks yang andal dan dapat mengenali sintaks-sintaks Python. Vim adalah contoh yang baik. Tutorial ini akan dibagi dalam langkah-langkah beserta penjelasannya.

1. Direktori untuk produk

Buatlah sebuah direktori dengan nama *Hello* di bawah direktori produk Zope (umumnya di `/usr/lib/zope/lib/python/Products`).

2. Definisi class produk

Ini adalah langkah mutlak yang harus dikerjakan sebaik-baiknya. Walaupun produk yang kita buat hanyalah produk yang sangat sederhana, kita tetap perlu mendefinisikan sebuah *class* yang baik. Simpanlah class ini ke dalam file *hello.py* di direktori *Hello* yang telah dibuat di langkah (1).

```
from OFS import SimpleItem
from Globals import DTMLFile

class hello(SimpleItem.SimpleItem):

    """
    produk hello
    """

    meta_type = 'Hello'

    def __init__(self, id, title, msg):
        """
        inisialisasi instance baru untuk Hello
        """
        self.id = id
```

```
self.title = title
```

```
self.msg = msg
```

```
index_html = DTMLFile('dtml/
index_html', globals())
```

```
manage_editHelloForm = DTMLFile('dtml/
manage_editHelloForm', globals())
```

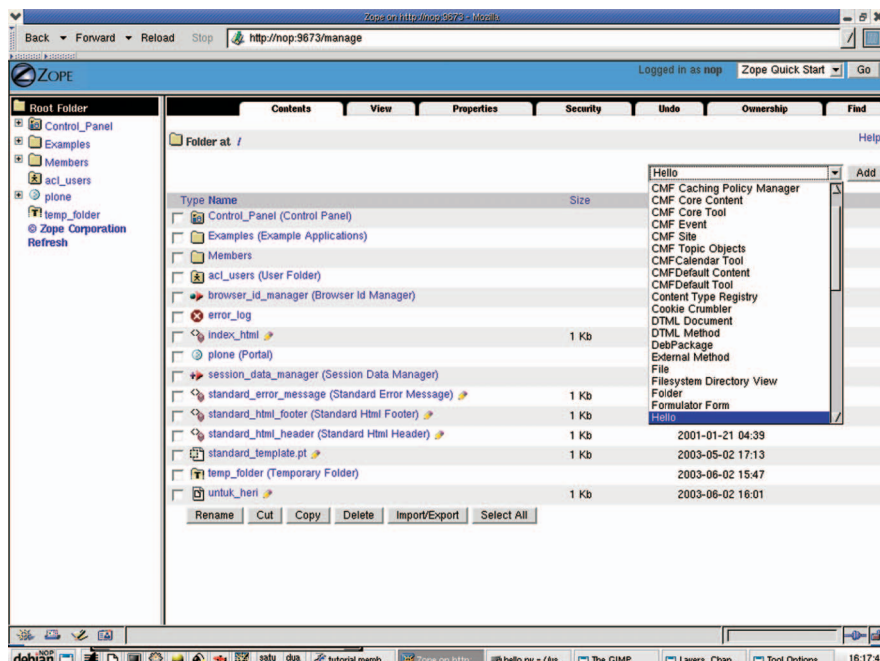
```
manage_workspace = DTMLFile('dtml/
manage_editHelloForm', globals())
```

```
manage_main = DTMLFile('dtml/
manage_editHelloForm', globals())
```

Kata kunci *meta_type* adalah kata kunci untuk nama produk kita yang akan tampil di *Add Product List Zope Management Interface*. Untuk itu, sebuah nama yang unik haruslah diberikan. Kemudian kita mendefinisikan fungsi `__init__()` yang berfungsi sebagai *constructor class*. Kemudian kita perlu juga untuk mendefinisikan sebuah tampilan default untuk objek yang instansiasi dari produk kita. Nama entiti default-nya adalah *index_html*.

Dalam hal ini, karena *index_html* akan menampilkan kode-kode HTML dan tag DTML, maka alih-alih menampilkannya di dalam kode class, kita akan menggunakan dokumen DTML yang terpisah dan diletakkan di direktori *dtml*, relatif dari direktori produk *Hello*. Demikian juga dengan *manage_editHelloForm*, *manage_workspace* dan *manage_main*. Tiga terakhir ini adalah entiti administratif yang menunjuk ke satu file DTML yang sama.

Kita perlu untuk menurunkan class kita dari class dasar yang akan sangat membantu, yaitu class *SimpleItem*. Tanpa menurunkan dari class lain, class *hello* yang kita buat tidak akan dapat digunakan seperti halnya produk Zope lainnya. Dan sebaliknya, dengan menurunkan dari *SimpleItem*, maka objek yang dibuat dari produk kita telah dapat di-cut dan di-paste ke *container* lain (contoh *container* yang paling baik adalah *Folder*), telah memiliki dukungan untuk bekerja dengan ZMI, memiliki dukungan FTP dan WebDAV, memiliki



Memilih Hello dari product list

dukungan untuk kepemilikan, dukungan *undo* dan lainnya. Luar biasa, bukan?

Berikut ini adalah beberapa hal yang perlu diperhatikan dalam membangun produk Zope:

- Semua class dan fungsi harus memiliki *documentation string*. Zope akan menolak semua produk yang tidak memiliki dokumentasi internal tersebut. Hal ini juga akan memudahkan *developer* lain yang membaca *source code* produk Anda.
- Pembuatan produk berarti pembuatan objek baru yang siap di instansiasi dan dipublikasikan. Zope sendiri akan melakukan publikasi objek (*Object Publishing*). Dengan demikian, apabila produk Anda nanti diinstansiasi dengan nama *tes_hello* misalnya, maka objek tersebut akan dipublikasikan oleh Zope dan dapat diakses melalui *web browser*. Semua hal adalah objek.
- Pada method `__init__()`, kita menuliskan kode `self.id = id`. Hal ini adalah mutlak karena setiap objek pada Zope akan memiliki atribut *id*.

3. Penambahan produk Hello oleh user

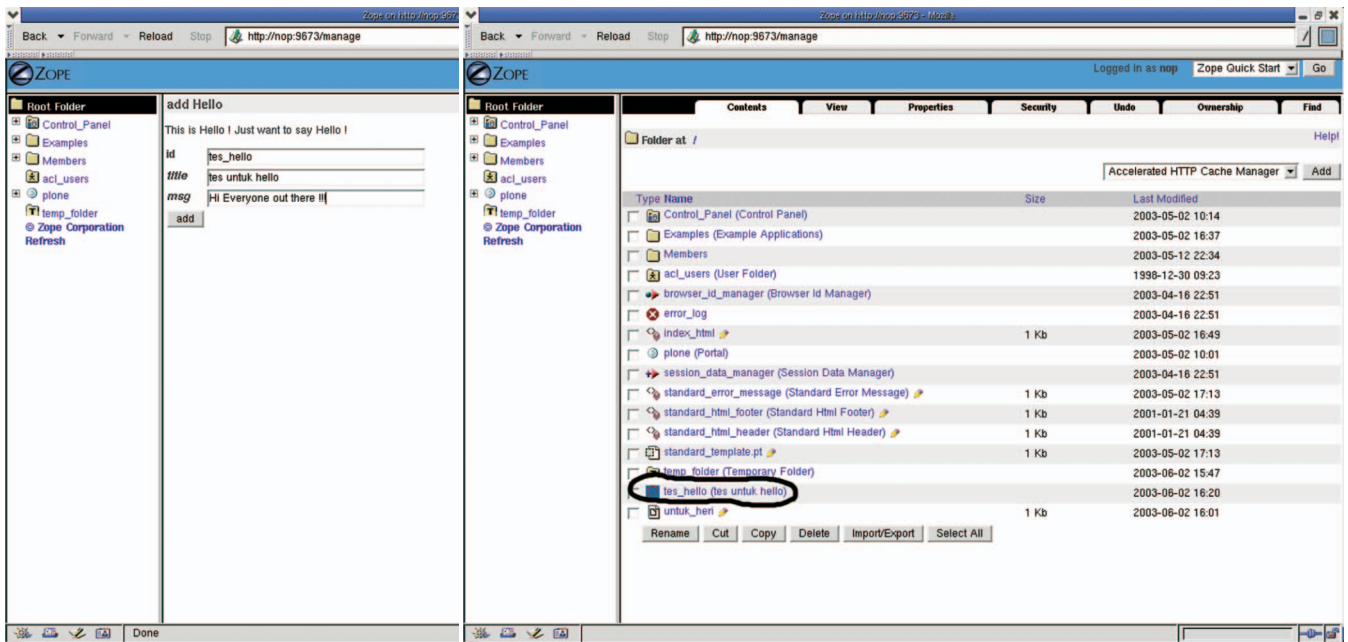
Pada produk-produk yang lain, kita akan disuguhkan sebuah *form* apabila ingin menambahkan produk tersebut ke dalam

suatu *container*. Demikian juga produk kita. Apabila user ingin menambahkan produk *Hello*, maka kita harus menampilkan suatu form kepada user. User kemudian dapat mengisi beberapa properti produk. Karena produk yang kita buat hanya memiliki tiga properti dengan nama *id*, *title*, dan *msg*, maka ketiga properti inilah yang dapat diatur oleh user. Tambahkan kode-kode berikut ini dalam file *hello.py* dan letakkan di luar definisi class *hello*.

```
manage_addHelloForm = DTMLFile('dtml/
manage_addHelloForm', globals())

def manage_addHello(self, id, title, msg,
REQUEST=None):
    """
    Response untuk manage_addHelloForm
    """
    self.setObject(id, hello(id, title, msg))
    if REQUEST is not None:
        return self.manage_main(self, REQUEST)
```

manage_addHelloForm adalah sebuah file DTML, sama seperti halnya *index_html*. Sementara fungsi *manage_addHello()* adalah *action* untuk form *manage_addHelloForm*. Method *setObject* akan mengurus penambahan objek ke *container*. Setelah penambahan berhasil, maka tampilan



▲ Form penambahan produk Hello

▲ Produk Hello telah ditambahkan ke root Zope

akan diarahkan kembali ke form *manage_main*.

4. Tambahan untuk class hello

Form *manage_editHelloForm* dapat digunakan untuk mengubah properti objek yang diinstansiasi dari produk kita. Dan tentunya kita pun perlu menyimpan segala perubahan yang dilakukan oleh user.

Zope sendiri datang bersama ZODB, sebuah objek database yang luar biasa. Segala objek dapat disimpan di dalam ZODB ini. Begitupun dengan perubahan yang kita lakukan. Tambahkan fungsi berikut ini ke dalam class hello:

```
def manage_editAction(self, title, msg,
REQUEST=None):
    """
    Menyimpan perubahan
    """
    self.title = title
    self.msg = msg
    self.p_changed = 1

    if REQUEST is not None:
        msgtouser = 'Hello %s updated'
        %self.id
        return self.manage_main(self,
        REQUEST, manage_tabs_message =
        msgtouser,)
```

Dengan memberikan nilai 1 kepada properti *_p_changed*, kita telah menyimpan segala perubahan pada objek tersebut.

Untuk memperindah tampilan layar edit produk kita yang didefinisikan di dalam file *manage_editHelloForm*, tambahkan baris-baris berikut ini persis di bawah kata kunci *meta_type*:

```
manage_options = (
{'label':'properties', 'action':'manage_
editHelloForm', 'help':('Hello',
'hello_edit.stx')}, {'label':'view',
'action':'index_html'},
)
```

Layar edit kita dilengkapi juga dengan fasilitas *Help* yang akan dibaca dari file *hello_edit.stx*. File dengan ekstensi *stx* dimaksudkan sebagai file Structured Text, sebuah teks terstruktur yang dapat di-*parse* oleh Zope dan ditampilkan dalam format yang indah.

Dengan demikian, berikut ini adalah definisi lengkap class hello yang kita miliki:

```
class hello(SimpleItem.SimpleItem):
    """
    produk hello
    """
```

```
meta_type = 'Hello'
```

```
manage_options = ({'label':'properties',
'action':'manage_editHelloForm',
'help':('Hello','hello_edit.stx')},
{'label':'view', 'action':
'index_html'},
)
```

```
def __init__(self, id, title, msg):
```

```
    """
```

```
    inisialisasi instance baru untuk Hello
```

```
    """
```

```
    self.id = id
```

```
    self.title = title
```

```
    self.msg = msg
```

```
index_html = DTMLFile('dtml/
index_html', globals())
```

```
manage_editHelloForm = DTMLFile('dtml/
manage_editHelloForm', globals())
```

```
manage_workspace = DTMLFile('dtml/
manage_editHelloForm', globals())
```

```
manage_main = DTMLFile('dtml/
manage_editHelloForm', globals())
```

```
def manage_editAction(self, title, msg,
REQUEST=None):
```

```
'''
Menyimpan perubahan
'''
self.title = title
self.msg = msg
self._p_changed = 1
if REQUEST is not None:
    msgtouser = 'Hello %s updated'
    %self.id
    return self.manage_main(self,
        REQUEST, manage_tabs_
        message=msgtouser,)
```

5. Inisialisasi untuk produk

Produk yang kita buat akan memasuki tahap-tahap penyelesaian dipandang dari sisi fungsionalisasi. Untuk saat ini, kita telah memiliki direktori Hello dan file hello.py di dalamnya. Sementara itu, kita masih memiliki utang untuk *index_html.dtml*, *manage_addHelloForm.dtml*, dan *manage_editHelloForm.dtml*. Selain itu, file help hello_edit.stx pun masih belum dibuat.

Untuk saat ini, Anda boleh melupakan kesemua utang kita tersebut. Adalah lebih penting untuk menyelesaikan file inisialisasi untuk produk kita. Buatlah file *__init__.py* dan letakkanlah di bawah direktori Hello:

```
import hello

def initialize(context):
    '''
    Inisialisasi produk Hello
    '''

    context.registerClass(
        hello.hello,
        constructors = (hello.manage_
            addHelloForm,
            hello.manage_addHello),
        icon='hello.png'
    )

    context.registerHelp()
    context.registerHelpTitle('Zope Help')
```

File *__init__.py* ini akan dibaca setiap kali Zope melakukan proses restart. Untuk itulah penambahan produk pada Zope mengharuskan Zope untuk melakukan restart. Pada file ini, kita mendaftarkan produk Hello dengan sistem Zope.

Buatlah sebuah file 16x16 pixel dan berilah nama hello.png. File tersebut, seperti yang terdefinisi di *registerClass()*, akan menjadi icon untuk produk kita. Kemudian, tak lupa, kita pun meminta Zope untuk mendaftarkan help kita.

Sampai saat ini, kita telah memiliki tiga file: *hello.py*, *__init__.py* dan *hello.png*. Dan pembuatan produk kita pun makin mendekati garis finish.

6. Pembuatan berbagai form

Saatnya untuk membayar hutang yang kita miliki satu per satu. Untuk form saja, kita memiliki 3 hutang: *index_html.dtml*, *manage_addHelloForm.dtml*, *manage_editHelloForm.dtml*. Pada kode di class hello, kita tidak menuliskan ekstensi *.dtml* secara eksplisit karena *DTMLFile()* telah mengetahuinya.

Pertama-tama, buatlah terlebih dahulu direktori dtml di bawah direktori Hello. Kemudian masuklah ke direktori dtml tersebut. Kita akan memulai dari *index_html.dtml*:

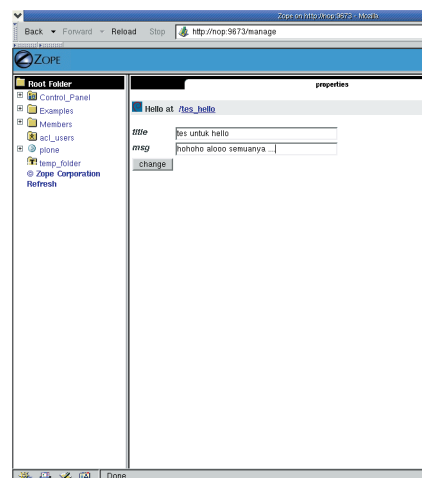
```
<dtml-var manage_page_header>

<h3 class="form-help">
<dtml-var id>

<dtml-if title>
(<dtml-var title missing="">)
</dtml-if>

says

</h3>
```



▲ Layar edit Hello

```
<hr noshade>

<dtml-if msg>
(<dtml-var msg missing='nothing'>)
</dtml-if>

<dtml-var manage_page_footer>
```

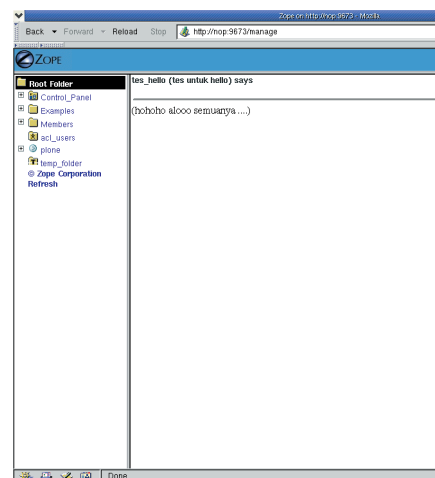
Kemudian, giliran hutang *manage_addHelloForm.dtml* yang dibayar:

```
<dtml-var manage_page_header>

<dtml-var "manage_form_title(this(),
    form_title = 'add Hello',
    help_product = 'Hello',
    help_topic = 'hello_edit.stx')
">

<p class="form-help">
This is Hello ! Just want to say Hello !
</p>

<form name=form action=manage_addHello
method=post>
<table border=0 cellpadding=0
cellpadding=2>
<tr>
<td valign="top" align="left">
<div class="form-label">
id
</td>
<td valign="top" align="left">
<input type=text name="id:string"
size=40> <br>
</td>
</tr>
```



▲ Index_html Hello

```

<tr>
  <td valign="top" align="left">
    <div class="form-optional">
      title
    </td>
    <td valign="top" align="left">
      <input type="text" name="title:string"
        size=40> <br>
    </td>
  </tr>

  <tr>
    <td valign="top" align="left">
      <div class="form-optional">
        msg
      </td>
      <td valign="top" align="left">
        <input type="text" name="msg:string"
          size=40> <br>
      </td>
    </tr>

    <tr>
      <td valign="top" align="left">
        <div class="form-element">
          <input class="form-element"
            type="submit" value="add">
          </td>
        </tr>

      </table>
    </form>

    <dtml-var manage_page_footer>

```

Dan inilah utang form kita yang terakhir: `manage_editHelloForm.dtml`:

```

<dtml-var manage_page_header>

<dtml-var manage_tabs>

<form name="form" action="."
method="post"> <br>

<table border=0 cellpadding=2>
<tr>
  <td valign="top" align="left"
class="form-optional">
  title

```

```

</td>
<td valign="top" align="left">
  <input type="text"
name="title:string" size="40"
value="<dtml-var title>">
  </td>
</tr>

<tr>
  <td valign="top" align="left"
class="form-optional">
    msg
  </td>
  <td valign="top" align="left">
    <input type="text"
name="msg:string" size="40"
value="<dtml-var msg>">
  </td>
</tr>

<tr>
  <td valign="top" align="left"
class="form-element">
    <input class="form-element"
type="submit" value="change"
name="manage_editAction:method">
  </td>
</tr>

</table>

</form>

<dtml-var manage_page_footer>

```

7. Pembuatan Help

Utang kita belum lunas. Masih ada beberapa hal yang perlu dikerjakan, salah satunya adalah pembuatan help. Di dalam form `manage_addHelloForm` dan file `hello.py` sendiri, kita mendefinisikan file `hello_edit.stx`. Sebenarnya, Anda dapat membuat file help sebanyak yang Anda inginkan. Anda bahkan dapat pula membuat help untuk API produk Anda. Karena topik pembuatan help sangatlah luas, maka artikel ini hanya akan membahas sampai help dalam format teks.

Buatlah direktori help di bawah direktori Hello dan masuklah ke dalam

direktori help tersebut. Dengan editor favorit Anda, ketikkan isi file `hello_edit.stx` berikut ini:

```

Hello

Just want to say Hi !

```

Karena kita menggunakan format STX, maka aturan penulisan STX haruslah dituruti, atau help Anda akan menjadi lebih susah untuk dibaca.

8. README.txt dan version.txt

Produk yang kita buat telah selesai 99,9% kalau tidak mau dikatakan selesai 100% untuk saat ini. Fungsi utamanya telah selesai kita kerjakan. Begitupun dengan hutang-hutang form dan help yang telah kita lunasi. Semuanya telah selesai. Tapi, ada baiknya kita melengkapi produk kita dengan `README.txt` dan `version.txt`. Kedua file ajaib tersebut akan menjadikan produk kita lebih informatif bagi Zope dan user.

Di bawah direktori Hello, isilah `README.txt` dengan teks berikut ini:

```

Hello

Just want to say Hi !

```

Dan isilah `version.txt` dengan teks berikut ini:

```

Hello-0.0.1

```

Kini produk yang kita buat telah selesai 100%. Walaupun secara fungsional produk kita tidak melakukan tugas apapun, tampilan dan kelengkapannya dapat disamakan dengan produk lain. Presentasi dengan file DTML pun telah menjadikan produk kita cukup modular. Untuk mengujinya, buatlah sebuah *instance* baru untuk produk Hello di root Zope. Berikanlah nama `tes_hello`. Objek yang baru Anda buat tersebut dapat diakses di `http://<ZOPE>/tes_hello`.

Saat ini, Hello adalah contoh produk sederhana. Anda dapat mengembangkannya menjadi jauh lebih berfungsi dan dapat digunakan oleh yang lain. Hari ini Hello, entah satu atau dua bulan ke depan. Selamat berkreasi!

Noprianto (noprianto@infolinux.co.id)