Welcome to session 17 InDesign CS/InCopy CS Plug-in Development training sessions.

---

Copyright 2004 Adobe Systems Incorporated. All rights reserved.

The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in this document. The software described in this document is furnished under license and may only be used or copied in accordance with the terms of such license.

Adobe, Adobe InCopy, and Adobe InDesign are trademarks of Adobe Systems Incorporated that may be registered in certain jurisdictions. Macintosh and Apple are registered trademarks, and Mac OS is a trademark of Apple Computer, Inc. Microsoft, Windows, Windows 95, Windows 98, Windows NT and Windows XP are registered trademarks of Microsoft Corporation. All other products or name brands are trademarks of their respective holders.
In Session 17, titled Cross Media, we will discuss the common use cases for cross-media related features primarily in InDesign.

We are purposefully limiting ourselves to “consumer” use cases, with the goal of automating existing features in InDesign. (We will provide various insights into how you can add custom features to InDesign, or “extend” the features of InDesign, however, that will not be the main focus of these sessions.)
This session will cover the InDesign Cross Media features from a plug-in developer’s standpoint. The Cross Media features in InDesign provide ways to exchange content data between InDesign and other applications, based on the eXtensible Markup Language (XML) file format.

In this session, we shall cover the following topics:

- Prerequisites from earlier sessions
- InDesign Cross Media Basics – a brief overview of what Cross Media features InDesign provides to the general user
- A highlight of the InDesign Cross Media architecture – where we will discover how the InDesign Cross media features coexist with the rest of InDesign features.
- A listing of Documentation and References available in the SDK and with this training session.
- And a recap of what we will have covered.
Prerequisites

- You must be comfortable with these topics before proceeding
  - Session 5: Document Structure
  - Session 11: Text Model
  - Cross media features are essentially add-ins to the document structure and text models

This slide shows a recommended list of topics for which you should already have a good understanding before proceeding.

Since the Cross Media subsystem is essentially a set of interfaces added into the document structure and text models, knowledge of how page item and text contents are stored in a document would greatly accelerate your understanding.

If you are not familiar with those two subsystems, please refer to the training sessions listed on the slide. In addition, here are some good SDK references to review:

[Page Items, Document Structure]
- InDesign Programming Guide: Document Structure chapter, Page Items chapters
- CodeSnippet: SnpInspectLayoutModel.cpp
- Plug-ins: Various plug-ins in the (SDK)/source/sdksamples folder, including CHMLFilter.

[Text]
- InDesign Programming Guide: Text chapters
- CodeSnippet: Snp*Text*.cpp
- Plug-ins: Various plug-ins in the (SDK)/source/sdksamples folder

All of the Adobe InDesign CS/InCopy CS Plug-in Development training session presentations are available for download at the ASN Partners site at http://partners.adobe.com/asn/indesign/sdk.jsp. To access this content, click the link for the exploded SDK, use your ASN Web Login to login, and then go to the “training” folders. You can also access the recordings for each of these sessions from the Q&A pages.
Let’s start with the fundamentals.
We’ll start with the discussion of why one would want to use XML with InDesign. Then, we will review how InDesign presents its XML features thru the user interface.
We will then touch upon a few side notes before we get into the architecture.
Why use XML with InDesign?

- "Network Publishing"
  - Publish content from one source to any device, anywhere
  - Repurpose content from various sources into InDesign

- Examples
  - Newswire feed in NewsML format drives copy layout in an InDesign or InCopy galley
  - Re-publish an InDesign galley via the web, various mobile protocols, or with a database

XML is a very flexible markup language, in that you can define your own tags. Well, a million software engineers can’t be wrong -- XML is one of the most widely used technologies in the software world today. A plethora of parsers, editors, style sheet generators, APIs, etc. are available all built around the XML-based standards. There are various XML-based standards that have been approved and/or accepted by many governing organizations, such as SVG, SOAP, NewsML, etc. and the acceptance of these standards continue to grow over time.

It’s because of these emerging standards that makes XML a wonderful medium to help InDesign communicate with other applications either running locally, over a network, or over the internet.

Adobe Systems corporate vision states that the company “helps people and businesses communicate better through its world-leading digital imaging, design, and document technology platforms for consumers, creative professionals, and enterprises.” Part of the vision is “Network publishing” (http://www.adobe.com/aboutadobe/pressroom/pressmaterials/networkpublishing/main.html), where a single source of information content is repurposed in many different ways so that the information can reach people at different locations, using different media types to consume the information. For instance, this information can originate from an InDesign page layout and be disseminated in different formats, or the information can originate from somewhere else (such as a web server or a news feed) and end up on an InDesign page layout. The InDesign Cross Media features were introduced to fulfill this vision.
How to use InDesign’s XML Features

- Manage tags with the Tag Panel
- Mapping tags to para/char styles and vice versa
- Assign tags to text and page items
- Assign attributes

Let’s examine what Cross Media features are provided with InDesign’s user interface.

- Tags as a way to identify elements
- Tag-to-Style Mapping, Style-to-Tag Mapping (Character and Paragraph Styles)
- Tagging page items and bodies of text, or elements
- Assigning attributes to tagged elements
- Importing a logical XML structure from an XML file into a document, and exporting the logical XML structure out to an XML file

Since this session focuses on the API, we won’t go into further detail regarding how to use these features.

If you require further assistance on how to use these features, please refer to the InDesign on-line help.
How to use InDesign’s XML Features (contd.)

- Export to XML (File >> Export…)

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Root>
  <MyFrameTag>
    <Tag1>This is a paragraph with the tag “Tag1”.</Tag1>
    <Tag2 MyAttr="MyAttrValue">This is a paragraph with the tag “Tag2”.</Tag2>
    <Tag3>This is a paragraph with the tag “Tag3”.</Tag3>
  </MyFrameTag>
</Root>
```

Shown on this slide is what an XML file that has been exported from an InDesign document (with tag markups) looks like.

The XML sample shown here is the result of exporting a text story that has been tagged with XML tags.

Every XML file begins with “<?xml and a version attribute, indicating the XML syntax version that the file complies with. Throughout the XML file are pairs of element begin and end tags.
### How to use InDesign’s XML Features (contd.)

- **Import from XML (File >> Import XML…)**
  - Select XML file to import, produce layout and drag Tagged Elements to layout, or
  - Have pre-assigned “root” level tag to pre-placed items, and then import
    - Autoflows content beneath “root” level tag

Upon importing an XML file into a new blank document, all you is a structural view on the left side of a document, shown in a tree view. The ImportXML doesn’t really serve a purpose by itself, until that logical structure represented by XML is associated with various content in a document.

There are two ways to correlate the logical structure to the document layout at import time.

1. You first import the XML file, then you produce the layout that you want to drag contents into. This way, the process is still dependent on the user’s ability to make connections between XML logical structure and the document content in an efficient way.

2. Have a document (or template) that has preplaced items with preassigned tags at the “root” level, then let the XML import make the connections between the logical structures underneath and the document content.

As you probably know, InDesign documents already have a hierarchical document structure. So you might be getting confused as to why there are now two different “trees” that could represent the document content…
SIDENOTE: Logical vs. Document Structure

- You can create a hierarchy that is different from the page item hierarchy

**Logical**

```
<Root>
  <Article>
    <Headline>…</Headline>
    <Body>
      <Photo src="…">
      …
```

**Document**

```
Spread
  Spread layer
  Spline item
  Multicolumn item
  Frame item
  Story
  Inline graphic
```

The main benefit of having a logical structure along side the existing document structure is in the flexibility, of being able to abstract out the document content in a way that makes sense in different contexts.

For instance, imagine an InDesign document containing a newspaper page layout. The InDesign document structure would be described as having a spread, a spread layer, and depending on the spread layer there would be numerous text frames (multicolumn item) that are indirectly associated with the news story components (headline, photo, caption, body text, etc.) There were be other spline items that provide the aesthetics of the page (borders). Also, on the same page would be advertisements represented as graphic items (e.g. PDF).

If one were to write a plug-in to reason about the content of that page, immediately one would discover the need for a way to describe what each one of the items on the page are. A multicolumn item could be a headline, a caption body text, etc. A page item could be part of a story, could be a picture, or just a headline. Data that describes the items, or meta data, would need to be associated somehow with each of the items.

That’s where the logical structure comes into play. You can associate the a “tag” (name of a meta data item) with a certain graphic frame (that could be a photo), or a certain story (NOTE: not the multicolumn item – as indicated by the big red X). The tagging can be done independent of the layout – even if the layout changes, the tags can move with the content.

It might be too “expensive” to have to create one page item for every thing you want to tag, so it would be useful if you can associate tags to portions of textual content, and furthermore, provide an semi-automated association. That led to the tagging of text stories and paragraph style mapping. (For example, you can have a Headline tag, a Subtitle tag, and Body tag all in the same text story.)

The XML features in InDesign provide the user with exactly that – the ability to abstractly describe a document in a flexible way.
A few sidenotes:

InDesign provides support for three different kinds of character encoding: UTF-8, UTF-16, and ShiftJIS.

ShiftJIS is primarily for Japanese users, however, since the ShiftJIS character set covers a smaller range of characters than Unicode, some InDesign text characters may not map out the internal Unicode characters correctly when exporting an XML file. For that reason, it is recommended that users try to use one of the Unicode encodings as much as possible.
Now, let’s take a detailed tour of the Cross Media architecture, from a development point of view.

We will uncover the Cross Media architecture staring from the most fundamental piece of the puzzle, and work our way upwards and outwards.

We will start with a description of the model-related elements in the Cross Media architecture, followed by selection suites, import/export and preferences.
Overview of the XML Model

- **XML model consists of three key elements**
  - Tags
  - Elements and attributes (and backing store)
  - Tag-Style Mappings
- **Associations are provided in**
  - Content-related bosses
  - Character/Paragraph Style bosses
  - The two workspace bosses
    - kDocWorkspaceBoss and kWorkspaceBoss

The XML model consists of the three key elements:
- Tags
- Elements and Attributes
- Tag-Style mappings

Let’s take a closer look at each one of these elements.

In reality, some of the preferences are also included as the XML model, however, they are either Import/Export related, or UI preference related. We will touch on those in later sections.
Tags

- Model Description
- Use cases
Tags are metadata labels we can put on InDesign document content.

Tags are represented as a kXMLTagBoss object, with IXMLTag being the characteristic interface that contains the name, and IPersistUIDRefData that contains the color of the tag marker. This is the color that is used when the tag markers are visible in the InDesign layout.

Collections of tags are accessible from the IXMLTagList interface, aggregated on both workspace bosses. IXMLTagList provides a set of enumeration methods on the set of tags.

The modification of tags happens through commands. However, one thing that is unique to the Cross Media API is that a vast majority of the API commands have a corresponding utility method in a UtilsBoss interface, such as IXMLTagCommands (shown on this slide). We’ll discover more later.
Bridge methods

- IXMLUtils::QueryXMLTagList(IDataBase*)
- IXMLTagList::GetTag(PMString), GetNthTag(int32)

Class Navigation

- `kDocWorkspaceBoss` (IXMBookmark)
- `kWorkspaceBoss` (IXMBookmark)

IXMLUtils is a very convenient kUtilsBoss interface, that offers the majority of XML related “bridge methods”. A bridge method is a convenient method that “bridges” from one boss to another. (NOTE: Remember that methods that start with the name Get***() usually do not increment reference counts, while those that start with Query***() do increment reference counts.)

IXMbookmark itself has a few bridge methods that help you find the tag you want.

The bottom part of this slide shows the class diagram – IXMLTagList is aggregated on both bosses, and you can arrive at the kXMLTagBoss (the tag) by using the IXMLTagList methods. Note that the kDocWorkspaceBoss contains the tag list for a specific document, and the kWorkspaceBoss contains the tag list which will be used as defaults when a new document is created.

NOTE: These UML class diagrams only show the essential interfaces. There are usually more interfaces on each boss class than what is depicted on the slide. Refer to the HTML-based reference documentation for each boss class, for more details.
[Use Cases: Tags]

- Iterating over the tags in a workspace is done by:
  - Querying for IXMLTagList on a workspace boss
  - Calling IXMLTagList::GetNumberOfTags and GetNthTag to get each tag’s UID
  - The UID corresponds to a kXMLTagBoss object, so you can query for IXMLTag.
Use Cases: Tags (contd.)

- Create a tag
  - IXMLTagCommands::CreateTag
- Delete a tag or all unused tags
  - IXMLTagCommands::DeleteTag (this replaces tag with another)
  - IXMLTagCommands::DeleteUnusedTags
- Set a tag name
  - IXMLTagCommands::SetTagName
- Save tags to file (XML)
  - IXMLTagCommands::SaveTagList
- Load tags from file (XML)
  - IXMLTagCommands::LoadTagList
- See SnpPerformXMLTags::Run

[Use Cases: Tags]
These use cases are very self explanatory. The IXMLTagCommands interface is very easy to use. If you need to look at a sample, please refer to the SnpPerformXMLTags::Run (SnpPerformXMLTags.cpp is in the {SDK}/source/sdksamples/codesnippets folder).
Elements and Attributes

- Model Description
- Use cases
Elements provide the connection between tags and InDesign document content. They are represented as XML elements. XML elements also contain can have a set of attribute names and values assigned. This is how one tag can be distinguished from other.

The relation between the tags and the content is stored in what’s called “XML Backing Store”. This separate boss is an abstraction of the XML related data for each content item. The backing store exists for the following content bosses:

- the document (kDocBoss),
- graphic/image items, and
- text stories.

Each of these content bosses aggregate IXMLReferenceData, which is an interface wrapper to XMLReference. XMLReference, which provides a reference to the backing store object…. (continue to next slide)
Model: Elements and Attributes (contd.)

- “XML Backing Store” represented by:
  - kXMLElementBoss, kTextXMLElementBoss, kXMLDocumentBoss, kXMLCommentBoss, kXMLPIBoss, kXMLDTDBoss

- Identified by IIDXMLElement
  - IIDXMLElement represents a structure tree, similar to IHierarchy (Key point: This is how you navigate)
  - NOTE: “IID” here does not mean “InterfaceID”… The “ID” in “IID” here means “InDesign”.

The XML backing store is presented as standalone bosses, as shown here. Each of these backing store bosses aggregate an API called IIDXMLElement. The IID is a bit misleading, as it means “Interface InDesign…”, not “Interface Identification”.

(XML backing store)
IIDXMLElement

Model: Elements and Attributes (contd.)

- **Bridge Methods**
  - IIDXMLElement* XMLReference::Instantiate(…)
    - NOTE: IIDXMLElement* is instantiated from IDataBase, therefore it is AddRef()ed!
  - IIDXMLElement* IXMLUtils::QueryDocElement(…)
  - IIDXMLElement* IXMLUtils::QueryRootElement(…)
  - IXMLUtils::GetActualContent(UIDRef& contentItem)
  - UID IXMLUtils::GetGraphicFrame(…)
  - ITextModel* IXMLUtils::QueryTextModel(…)
  - XMLReference IXMLUtils::GetStoryXMLReference(…)

Again, IXMLUtils provides quite a few useful bridge methods.

One way to find out if a content boss has any XML elements associated with it is to see if you can use an XMLReference to instantiate it’s backing store, using XMLReference::Instantiate(). If you get a nil ptr, there’s no element associated with the content.

Also, refer to the xmedia.pdf technote for some how-to’s and tricks involving IXMLUtils.
Model: Elements and Attributes (contd.)

- **Class Navigation**
  - Start from the content item or root, get the XMLReference, then instantiate the backing store (IIDXMLElement)

You can go from the content item to its XML backing store, and back.

The key point in this slide is, no matter where you are in the XML hierarchy, or what type of object is being tagged (document root, page item, text story), the navigation between the content item and the corresponding XML backing store is the same.

The existence of these interfaces on a boss indicate what they are:
- If IXMLReferenceData: This boss is a content boss.
- If IIDXMLElement: This boss is a backing store boss.
[Use Cases: Elements and Attributes]

• Getting the “root” XML element can be done by calling IXMLUtils::QueryDocElement. This is the starting point of the logical structure, as viewed on the logical structure view of the document window (left pane).

• Iterating XML elements in the documents logical structure is just a matter of visiting all of the children elements using IIDXMLElement. For each element you visit, you get an XMLReference, which gives you access to the UID of the backing store for that particular element. The code snippet does it in two ways:
  • Starting from a document structure item (IHierarchy) – see note below.
  • Starting from the root XML element of the document.

NOTE about traversing the XML structure from a document structure item:
IHierarchy is used to visit children in the document structural hierarchy. If an item is a text frame, this also looks to see if there is a text story associated with the frame, since the text frame is never tagged, it’s the text story that is tagged.
Use Cases: Elements and Attributes (contd.)

- Change XML elements/attributes for a page item
  - Element
    - If an XMLReference exists, call IXMLElementCommands::SetElement
    - If an XMLReference does not exist, call IXMLElementCommands::CreateElement
  - Attribute
    - If attribute exists, add call IXMLAttributeCommands::SetAttributeName and/or SetAttributeValue
    - If attribute does not exist or to add one, call IXMLAttributeCommands::CreateAttribute
  - See SnpPerformXMLElements::TagFrameElement

[Use Cases: Elements and Attributes]

• Changing the XML element for a page item can be done by calling either IXMLElementCommands::SetElement (if an XMLReference already exists for the page item), or IXMLElementCommands::CreateElement (if an XMLReference does not exist for the page item).

• Changing the XML attribute for a page item can be done by calling either IXMLAttributeCommands::SetAttributeName and SetAttributeValue (if an attribute exists) or IXMLAttributeCommands::CreateAttribute. Note that some attributes cannot be removed.
[Use Cases: Elements and Attributes]

- To change XML elements/attributes for text, the APIs are similar, but an entire story must be tagged first.
- To delete an XML element, call IXMLElementCommands::DeleteElement or RangeDeleteElement.
- To delete an XML attribute, call IXMLAttributeCommands::DeleteAttribute.
Mapping Tags and Styles

- Model Description
- Use cases
Earlier, we discussed the usefulness in mapping text styles (paragraph styles) to tags so we can provide an efficient logical structure to a body of text. This part of the model allows just that – mapping tags and styles to each other. Tags to styles, and styles to tags.

One interesting thing about mapping tags to styles. Say you have some text already tagged with elements. If you map styles to tags, and if you had some styles already applied to the text that has tags, then the newly mapped style would override the existing style. (Mapping tags to styles is less intrusive.) So when you map styles to tags, you want to do this in a template document, or when the document is still empty and brand new.

Again, the mappings in both directions are stored in the workspace bosses (kDocWorkspaceBoss and kWorkspaceBoss).

As a sidenote:

How do you get the paragraph styles?

You can get them via character and paragraph style name table in workspace bosses, using the IStyleNameTable interface (IID_ICHARSTYLENAMETABLE and IID_IPARASTYLENAMETABLE, respectively). This leads you to kStyleBoss, the boss for the actual style. The interface with style info is IStyleInfo.
Bridge Methods
- IXMLUtils::QueryXMLStyleToTagMap(…)
- IXMLUtils::QueryXMLTagToStyleMap(…)

Class Navigation

Again, IXMLUtils provides a few bridge methods that help you find the mapping interfaces, and once you get them, you can use the bridge methods on the mapping interfaces to find the tag (kXMLTagBoss) and style bosses (kStyleBoss).
Use Cases: Mapping Tags and Styles

- **Get the style-to-tag mapping list**
  - Call IXMLUtils::QueryXMLStyleToTagMap
    - Returns IXMLStyleToTagMap
    - See SnpPerformXMLTagAndStyle::ReportStyleToTag

- **Get the tag that is mapped to a style**
  - Call IXMLStyleToTagMap::GetTagMappedToStyle
    - Returns UID of kXMLTagBoss

[Use Cases: Mapping Tags and Styles]
These are all very self-explanatory. The code snippet SnpPerformXMLTagAndStyle is a good reference.
[Use Cases: Mapping Tags and Styles]

These are all very self explanatory. The code snippet SnpPerformXMLTagAndStyle is a good reference.

- **Get the tag-to-style mapping list**
  - Call IXMLUtils::QueryXMLTagToStyleMap
    - Returns IXMLTagToStyleMap
  - See SnpPerformXMLTagAndStyle::ReportTagToStyle

- **Get the style that is mapped to a tag**
  - Call IXMLTagToStyleMap::GetStyleMappedToTag
    - Returns UID of kStyleBoss (see Session 11: Text Model)
### Use Cases: Mapping Tags and Styles (contd.)

- **Map a style to a tag**
  - Assuming styles and tags are already created, call `IXMLMappingCommands::MapStyleToTag`
  - See `SnpPerformXMLTagAndStyle::MakeStyleToTagMap`

- **Map a tag to a style**
  - Assuming tags and styles are already created, call `IXMLMappingCommands::MapTagToStyle`
  - See `SnpPerformXMLTagAndStyle::MakeTagToStyleMap`

---

[Use Cases: Mapping Tags and Styles]

Again, these are all very self-explanatory. The code snippet `SnpPerformXMLTagAndStyle` is a good reference.
[Use Cases: Mapping Tags and Styles]

If you have just created or modified a style-to-tag map, and want to automatically tag all of the content on a document or a specific story, you can call the IXMLElementCommands::CreateStyleToTagElements method. See HTML-based reference documentation on this interface method for more details on the parameters.
Import/Export and Preferences

- Model Description
- Use cases
Model: Import/Export and preferences

- Supplies implementations behind “File >> Import XML” and “File >> Export”
- Import Preferences
  - IXMLImportOptions (kDocWorkspaceBoss and kWorkspaceBoss)
- Export Preferences
  - IXMLExportOptions (kDocWorkspaceBoss and kWorkspaceBoss)
  - IXMLExportSessionOptions (kWorkspaceBoss only)

Importing and exporting are perhaps the most interesting part of the Cross Media API – they allow for Network publishing to happen!

From the InDesign user interface, users can import XML files using File >> Import XML, and export XML files using File >> Export (and select XML as the file type (or “formatName”, as we discussed in Session 14: Service Providers.). We’ll discuss what API correspond to those menu actions on the next slide.

First, let’s discuss where the Import/Export options are stored.

The import options, identified by the IXMLImportOptions interface, are aggregated in both the kDocWorkspaceBoss and kWorkspaceBoss. This interface represents the radio buttons (Replace or Append content) and the check box (that specifies if you want to import at the selected XML structure tree view node) on the Import XML Dialog.

There are two different kinds of export options. The IXMLExportOptions interface is aggregated on both the kDocWorkspaceBoss and kWorkspaceBoss, just like the IXMLImportOptions. The IXMLExportSessionOptions interface, which interface to session-specific XML export options, is only aggregated on kWorkspaceBoss. These interfaces collectively represent the data on the Export XML Dialog (the dialog that appears after you select an XML file to export). If you delete your InDesign preferences file, and have never set any of the IXMLExportSessionOptions since that time, the IXMLExportSessionOptions will show an empty string for the preferred browser path. NOTE: The only way to have a browser path set is to open the Export XML Dialog, or to set it using a command (see next slide).
[Use Cases: Import/Export]

- To modify the import preferences, process `kChangeXMLImportOptionsCmdBoss`.
- To import an XML file, process the `kImportXMLFileCmdBoss`. (NOTE: ImportXML is not implemented as an import provider, as an InDesign XML file can refer to several page items.)
[Use Cases: Import/Export]

To modify the export preferences, process `kChangeXMLExportOptionsCmdBoss` (workspace specific) and `kChangeXMLExportSOptionsCmdBoss` (kWorkspaceBoss only).

To export XML from a selection, use the XML export provider.

NOTE: Since exporting XML doesn’t involve a model change (unless you are changing preferences at the same time), there isn’t a command to do the exporting. However, if you know you are going to export to an XML file format, you can find one of the Export Providers that support XML export (by formatName “XML”). That Export Provider is `kXMLExportProviderBoss`, which is a ServiceProvider that services `kExportProviderService` and aggregates an `IExportProvider` implementation. You can use the `IK2ServiceRegistry` to find this export provider, then call the methods on `IExportProvider` to do the export. (Also see the codesnippet called `SnpExportSelection.cpp` – provided with Session 14: Service Providers – which uses a generic technique to find a service provider.)
### Use Cases: Import/Export and Preferences (contd.)

- **Exclude DTD reference upon XML export**
  - Call IXMLExportOptions::SetExcludeDTD when setting export options

- **Enable copying of tagged image files to a specified subdirectory**
  - Call IXMLExportOptions::EnableCopyOriginals when setting export options
  - **NOTE:** The images subdirectory name is hardcoded, therefore it cannot be modified
  - See SnpPackageForGoLive::SetExportOptions

---

[Use Cases: Import/Export]

Among the export options, these settings probably require further explanation.

- You can exclude a reference to a DTD during export by calling IXMLExportOptions::SetExcludeDTD(kTrue) when you set the workspace-specific export options (see previous slide). This prevents the DTD from being referenced in the generated XML file.

- You can also enable the copying of image files to the “images” folder (where the XML file will be written) by setting the IXMLExportOptions::EnableCopyOriginals to kTrue when you set the workspace-specific export options (see previous slide). The “images” subfolder name is hardcoded and can’t be changed.
XML-related Selection Suites
XML-related Selection Suites

- Performs tasks on current selections
- IXMLNodeSelectionSuite (on selected node)
  - kAbstractXMLSelectionBoss
- IXMLStructureSuite (on selected structure)
  - kIntegratorSuiteBoss, kXMLStructureSuiteBoss
- IXMLTagSuite (selected tag)
  - kIntegratorISuiteBoss, kXMLStructureSuiteBoss,
    kTableSuiteBoss, kLayoutSuiteBoss, kTextSuiteBoss,
    kGalleyTextSuiteBoss

Selection suites make the selection-based actions much easier to program. This is done by an intricate system based on a whole hierarchy of selection bosses, and a collection of suite interfaces. (See the Selection technote {SDK}/docs/guides/selection.pdf for more details)

Suite interfaces are what make the selection-based actions easy, with code like this:

```c++
InterfacePtr<IXMLTagSuite> tagSuite((IXMLTagSuite*)Utils<ISelectionUtils>()-
>QuerySuite(IXMLTagSuite::kDefaultIID));
If (tagSuite && tagSuite->CanUnTag())
    status = tagSuite->UnTag();
```

It doesn’t matter what is selected – the only thing we need to know is, can we do the action we want to do on whatever is currently selected.

There are three different XML selection suite interfaces offered.
- IXMLNodeSelectionSuite: perform operations on the selected node in the XML structure view (select/deselect element, etc.)
- IXMLStructureSuite: perform operations on the structure in the XML structure view (add/delete/replace elements or attributes, etc.)
- IXMLTagSuite: perform operations on the XML tag in the Tags panel

The bosses they are aggregated on indicate the type of selected object selected.

Note:
- k*SuiteBoss is the base boss class for k*SelectionBoss.
Notification
Notification

- What commands are available? To which boss's ISubject do I need to attach observers? What protocol should I observe?
  - …when an XML Element has been modified
  - …when an XML Tag has been modified
  - … when an XML tag and style mapping has been modified
  - … how to get details on what was changed
  - Answers are all in XMediaCommandReference.xls

So far, whenever we talk about modifying XML model data, we mentioned the use of command wrapper utility interfaces, such as IXMLElementCommands and IXMLTagCommands. The methods in such utility interfaces all process a command behind the scenes.

Knowing which command gets called for each XML model change is important for the following reasons:

• You may need to observe a specific type of model change, therefore you would need to know the command class ID, the protocol, and some details about what’s on the item list for the command.

• Knowing that some commands communicate their changes in a slightly different way is important when writing selection suite extensitions (purposely misspelled).

This would be a good time to take a look at the XMediaCommandReference.xls, located in the same folder as the PDF version of this presentation.

The XMediaCommandReference.xls Excel file contains the following information about each command:

• Name, category
• Info for use with IObserver (notification subject boss, changedBy, Protocol)
• Info for use as a command client (itemList in/out, interfaces)
• The utility interface that wraps the command
Notification (contd.)

- All XML structure changes on a document are notified via a single ISubject
  - The ISubject of the document’s backing store is where XML structure changes are notified
  - Use IXMLUtils::QueryDocElement() to get the document’s backing store, and query for the ISubject.
  - Protocol = IID_IIDXMLELEMENT
- XML import notification is done via ISubject of kDocBoss using protocol = IID_IIMPORTMANAGER

All XML structure changes (tagging elements, adding attributes, etc.) are notified via ISubject of the document’s backing store, using protocol IID_IIDXMLELEMENT.
Notification (contd.)

- Obtain the XML element changed by a command
  - Usually there is some data interface or item list that refers to the backing store object that was modified
    - You can get the ICommand* by typecasting void* changedBy, then query for a command data interface

To obtain the XML element changed by a command, you can typecast the void* changedBy parameter to an ICommand*, which gives you a pointer to the command. (This is a commonly used technique whenever observing changes by most commands.) Once you get a pointer to the command, you can call ICommand methods, such as GetItemListReference, or even query for other interfaces on the same command boss. Usually, that should give you a reference to the what was modified, whether it is an XMLReference or a UID of some content item.
An XML-based workflow

- Introduction to the XDocBookWorkflow plug-in
Putting it all together…

- What you do with XML is the key
  - A common developer problem?
    - Importing in an XML file (generated somewhere else), and
    - Creating layout items on the fly
  - XDocBookWorkflow: An XML-based workflow
    - {SDK}/source/sdkexamples/xdocbookworkflow
      - The examplefiles folder contains .indt files and images for use with this sample
    - {SDK}/source/sdkexamples/xdocbookworkflowui
      - User interface plug-in
      - NOTE: An example of model-view separation
    - See this plug-in’s design document for more info

We expect that customizing the import/export workflow will be the most popular 3rd party use-case. So let’s take a look at how one sample plug-in accomplishes this.

Take a look at the design document for this plug-in (in the HTML-based reference documentation, click on the “Sample plug-ins” link at the top of the page, then go to the link for “XDocBookWorkflow”). The Use section describes how to use this plug-in.

This plug-in relies on a few setup items:
• The InDesign template to use needs to be preset.
• The images folder needs to be preset.

These are not a general requirements of XML import, it’s just the way that this plug-in has been implemented to keep style and image management simple.
Let’s see how this plug-in works.

One of the key things offered in this plug-in is the XML import workflow automation provided by the XDocBookWorkflow plugin is the augmentation of the ImportXMLFileCmd from the import provider implementation, XDocBkImportProvider::ImportThis.

Up to the point of importing the XML file, there is nothing unusual. After this is where this plug-in adds value.

• XDocBkImportProvider::ImportThis calls XDocBkFacade::ConvertXMLToGraphics to create inline graphics.
• XDocBkImportProvider::ImportThis calls XDocBkFacade::ConvertXMLToTables to create tables.
• XDocBkImportProvider::ImportThis calls XDocBkFacade::RemapStyles which processes the kXMLApplyTagToStyleMappingCmdBoss command to map XML tags to corresponding paragraph/character styles.

You can study each step in more detail.
This session provided a brief overview of what InDesign’s Cross Media features are available, and what they mean to all of us as plug-in developers. Let’s review the tools that we have at our disposal in the SDK.
Here is a list of documents and samples that you can refer to while working with this training session.

The HTML-based reference documentation is of particular interest to developers, since it includes a detailed references of our API and boss classes.
Summary

- Prerequisites: Review from earlier sessions
- InDesign Cross Media Basics
- Cross Media Architecture
  - Model Descriptions and Use Cases
    - Tags, Elements and Attributes, Style Maps
    - Import/Export and preferences
  - Selection Suites
  - Notification
- Documentation, References

Summary of this session.

If time allows, this is a good time for Q&A.
Thanks for attending this session. We wish you good luck in your plug-in development endeavors!